TEKNILLINEN KORKEAKOULU
Sähkötekniikan osasto

Ilari Aarnio

TIETOKANNAN KONVERSIO-OHJELMIEN KUVAUSKIELI JA ESIKÄÄNTÄJÄ DX 200 -JÄRJESTELMÄSSÄ

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi diplomi-insinöörin tutkintoa varten Espoossa 1.6.1993

Työn valvoja Heikki Saikkonen

Työn ohjaaja Matti Tikkanen

19142

TKK SÄHKÖTEKNIIKAN
OSASTON KIRJASTO
OTAKAARI 5 A
02150 ESPOO

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ

Tekijä: Ilari Aarnio

Työn nimi: Tietokannan konversio-ohjelmien kuvauskieli ja esikääntäjä DX 200 -järjestelmässä

Päivämäärä: 27.5.1993 Sivumäärä: 75

Osasto: Sähkötekniikka

Professuuri: Tietojenkäsittelyoppi

Työn valvoja: Heikki Saikkonen

Työn ohjaaja: Matti Tikkanen

DX 200 -järjestelmän puhelinkeskuksen ohjelmisto muodostaa ohjelmistopaketin. Puhelinkeskuksien ohjelmistopaketteja vaihdetaan uusien ominaisuuksien lisäämiseksi ja vikojen korjaamiseksi. Ohjelmistopaketin vaihdon yhteydessä vanhan ohjelmistopaketin tiedostot ja tietokannat on konvertoitava uuden ohjelmistopaketin edellyttämään muotoon. Näitä konversioita suoritettavia ohjelmia kutsutaan konversio-ohjelmiksi ja ne on tapauskohtaisesti ohjelmoitu C- tai PL/M-kielillä.

Diplomityössä on kehitetty tietokantojen konversio-ohjelmille kuvauskieli. Sen avulla konversio-ohjelmat voidaan kuvata korkeammalla abstraktiotasolla ja käyttäen tietokannan käsitteitä. Kuvauskielelle on esitetty kaksi vaihtoehtoa: deklaratiivinen ja proseduraalinen kieli. Deklaratiivisessa kuvauskielessä konversiotoimenpiteet on ennalta määritelty ja sisällytetty kielen syntaksiin. Proseduraalisessa kuvauskielessä konversiotoimenpiteiden kuvataan ohjelmoimalla toimenpiteet toteuttava konversiolohko. Näistä vaihtoehdoista valittiin kuvauskieleksi proseduraalinen kieli sen laajemman käyttöalueen ja DML-kielen kanssa yhtenevien lauserakenteiden vuoksi.

Kuvauskielen toteuttava esikääntäjä tuottaa konversio-ohjelmien kuvauksista C-kielisen konversio-ohjelman lähdekoodin. Esikääntäjän tavoitteet määräsivät toisaalta esikääntäjän toteutuksen tavoitteet ja toisaalta esikääntäjällä tuotettujen konversio-ohjelmien tavoitteet. Esikääntäjän tavoitteista tärkein on siirrettävyys, sillä esikääntäjän käyttöympäristöinä tullaan käyttämään ainakin VMS- ja Unix-ympäristöjä. Tuotettujen konversio-ohjelmien asettamista tavoitteista tärkein on suorituskyky, josta johtuen esikääntäjän on huomioitava suorituskykynäkökohdat koodintuotossa.

Avainsanat: konversio, kääntäjä, ohjelmointikieli, tietokanta

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF THE MASTER'S THESIS

Author: Ilari Aarnio

Name of the thesis: Description language and precompiler for database conversion programs in the DX 200 system

Date: May 27th, 1993

Number of pages: 75

Faculty: Electrical Engineering

Professorship: Computer Science

Supervisor: Heikki Saikkonen

Instructor: Matti Tikkanen

Software package is a collection of software used in DX 200 telephone exchange. In order to fix errors and introduce new features the software packages are changed. In the process of software package change the contents of files and databases of older software package have to be converted to the form required by newer software package. These conversions are executed by conversion programs that have been programmed in C or PL/M.

The purpose of the thesis is to develop a description language for conversion programs. The language should use higher level of abstraction and utilize database concepts. Two alternatives for language is presented. One is declarative and the other is procedural. The syntax of the declarative language contains predefined conversion procedures whereas the syntax of procedural language contains more general statements. The procedural alternative was chosen to be the description language of conversion programs because of its wider range of use and similarities with DML-language.

The description language precompiler generates source code of conversion programs in C. Two types of requirements was set. Some requirements concern the precompiler implementation itself while the other requirements concern the conversion programs generated by the precompiler. The most important implementation requirement is portability. The precompiler will be used at least in VMS and UNIX environments. The most important requirement of generated conversion programs is their performance. In order to fulfil this requirement the precompiler must favour performance in code generation.

Keywords: conversion, compiler, programming language, database

ALKULAUSE

Haluan esittää kiitokseni Matti Tikkaselle, Maaret Karttuselle ja Jukka Aakulalle monissa heidän kanssaan käymissäni palavereissa saamistani neuvoista ja ohjauksesta. Ilman heidän asiantuntemustaan tässä diplomityössä esitetyn kuvauskielen ja esikääntäjän tekeminen ei olisi ollut mahdollista.

Osa kuvauskielen syntaksista on kehitetty muiden ihmisten toimesta. Määrittelyosuus on pieniä poikkeuksia lukuunottamatta Peter Hjortin ja Erkki Ruohtulan diplomitöissään [1] ja [2] esittelemän TNSDL-kielen määrittelysyntaksin mukaista. Konversiolohkon lauseiden syntaksi on suurelta osin Matti Tikkasen kehittämän TDL-kielen [3] mukaista.

Helsingissä 27.5.1993

SISÄLLYSLUETTELO

KÄSITTEET

Alustus

Alustuksilla tarkoitetaan objektikokoelmiin, hakemistoihin, relaatioihin ja laskennallisiin suhteisiin kohdistuvia muutoksia, jotka tehdään yhden tietokantailmentymän sisällä.

DDL

DDL (Data Definition Language) on loogisen tason tiedonmäärittelykieli. TDL-kielen looginen osajoukko, jolla kuvataan tietokantakaavio.

DML

DML (Data Manipulation Language) on tiedonkäsittelykieli. TDL-kielen looginen osajoukko, jolla ohjelmoidaan tietokantaproseduurit.

DX 200

Nokia Telecommunications Oy:n digitaalinen puhelinkeskusjärjestelmä.

FDL

FDL (Physical Data Definition Language) on fyysinen tason tiedonmäärittelykieli. TDL-kielen looginen osajoukko, jolla kuvataan tietokannan toteutusmäärittely.

Konversio

Konversioilla tarkoitetaan objektikokoelmiin, hakemistoihin, relaatioihin ja laskennallisiin suhteisiin kohdistuvia muutoksia, jotka tapahtuvat eri ohjelmistopakettien välillä yhdestä tietokantailmentymästä toiseen.

Laskennallinen suhde

Tietokantakäsite, jolla kuvataan kahden objektikokoelman ja yhden suhdetta kuvaavan attribuutin riippuvuus. Laskennallinen suhde on objektirelaation erikoistapaus, jossa yksi kolmipaikkaisen objektirelaation objektikokoelmista on korvattu tilaasäästävällä attribuutilla.

Metakieli

Kielen muodostussääntöjen esittämiseen käytettävä kieli.

Objekti

Tietokantakäsite, joka muodostuu tyypistä ja yksilöllisestä objektitunnisteesta. Jokainen objektin ilmentymä kuuluu johonkin objektikokoelmaan.

Objektihakemisto

Tietokantakäsite, jolla objektien hakua objektikokoelmasta voidaan tehostaa. Objektihakemiston hakuavain muodostuu yhdestä tai useammasta objektin tyypin kentästä.

Objektikokoelma

Tietokantakäsite, jolla ryhmitellään samaa tyyppiä olevia objekteja.

Objektirelaatio

Tietokantakäsite, jolla kuvataan kahden tai useamman objektikokoelman välisiä riippuvuuksia.

TDL

TDL (Telenokia Database Language) on tietokantakieli, jolla kuvataan DX 200 -järjestelmän tietokannat ja niiden käsittely.

TNSDL

TNSDL (Telenokia Specification and Description Language) on Nokia Telecommunications Oy:ssä kehitetty määrittely- ja ohjelmointikieli.

1. JOHDANTO

Diplomityö käsittelee DX 200 -järjestelmän tietokantojen konversio-ohjelmien kuvaamiseen kehitettyä kuvauskieltä ja esikääntäjää. Niiden kehitystyö on tapahtunut Nokia Telecommunications Oy:n keskusjärjestelmien tutkimus- ja tuotekehitysyksikössä.

Lähtökohtana diplomityölle on ollut tarve kehittää korkean abstaktiotason kuvauskieli tietokantojen konversio-ohjelmille, jotka ennen on ohjelmoitu tapauskohtaisesti C- tai PL/M-kielellä. Tietokantakonversioiden tarve on jatkuvasti lisääntymässä tietokantojen korvatessa yksittäiset tiedostot tiedonhallinnan välineinä.

Tässä dokumentissa on selostettu yleisellä tasolla DX 200 -järjestelmän muodostava laitteisto ja ohjelmisto. Tarkemmin on käsitelty järjestelmään sisältyvä monitietokantajärjestelmä ja sen kuvaamiseen käytetty TDL-tietokantakieli. Näiden lisäksi dokumentissa on kerrottu, milloin tietokantakonversioita tarvitaan ja miten niitä suorittavia konversio-ohjelmia käytetään.

Kuvauskielen suunnittelun lähtökohdat ja tavoitteet on selvitetty. Tavoitteita on asetettu sekä toiminnallisesta että ohjelmointiteknisestä näkökulmasta. Toiminnallisiin tavoitteisiin sisältyvät konkreettiset käyttötapaan ja suorituskykyyn liittyvät seikat. Ohjelmointitekniset tavoitteet ovat sen sijaan yleisluonteisempia ohjelmointikielten suunnitteluun liittyviä tavoitteita.

Kuvauskielelle on esitetty kaksi vaihtoehtoa: deklaratiivinen ja proseduraalinen kieli. Deklaratiivisessa kuvauskielessä konversiotoimenpiteet on ennalta määritelty ja sisällytetty kielen syntaksiin. Proseduraalissa kuvauskielessä konversiotoimenpiteiden kuvaustapa on ohjelmointipainotteinen. Kuvauskielten etuja ja haittoja on arvioitu, ja valinta kuvauskielten välillä on tehty.

Dokumentti on kuvauskielipainotteinen, sillä sen laatiminen oli diplomityön haastavin osuus. Dokumentin loppuosassa käsitellään kuvauskieltä varten laadittavan esikääntäjän tavoitteita, toteutusta ja toimintaa. Tavoitteet sisältävät itse esikääntäjäohjelmalle asetettujen tavoitteiden lisäksi esikääntäjällä tuotettujen konversio-ohjelmien tavoitteita. Toteutuksessa käydään läpi esikääntäjän toteutusympäristö ja sen tarjoamat kehitystyökalut sekä toteutuksen vaiheistus. Lopuksi käsitellään esikääntäjän toimintaan liittyviä seikkoja, kuten käyttöliittymää ja esikääntäjän osuutta konversiokuvauksen saattamisessa ajokelpoiseksi konversio-ohjelmaksi.

2. DX 200 -JÄRJESTELMÄ

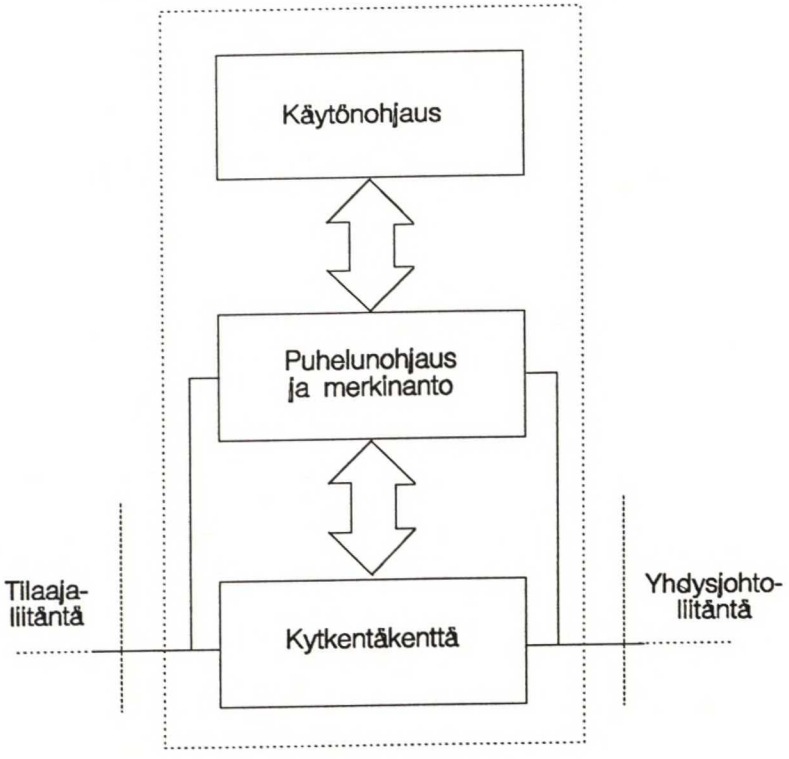
2.1. YLEISTÄ

DX 200 -järjestelmä on Nokia Telecommunications Oy:n digitaalinen tietokoneohjattu puhelinkeskusjärjestelmä. Se on hajautettu sekä laitteiston että ohjelmiston osalta.

2.2. LAITTEISTOARKKITEHTUURI

DX 200 -järjestelmän laitteisto voidaan jakaa karkeasti kolmeen osaan: puhelunohjaus-, kytkentä- ja käytönohjausosaan.

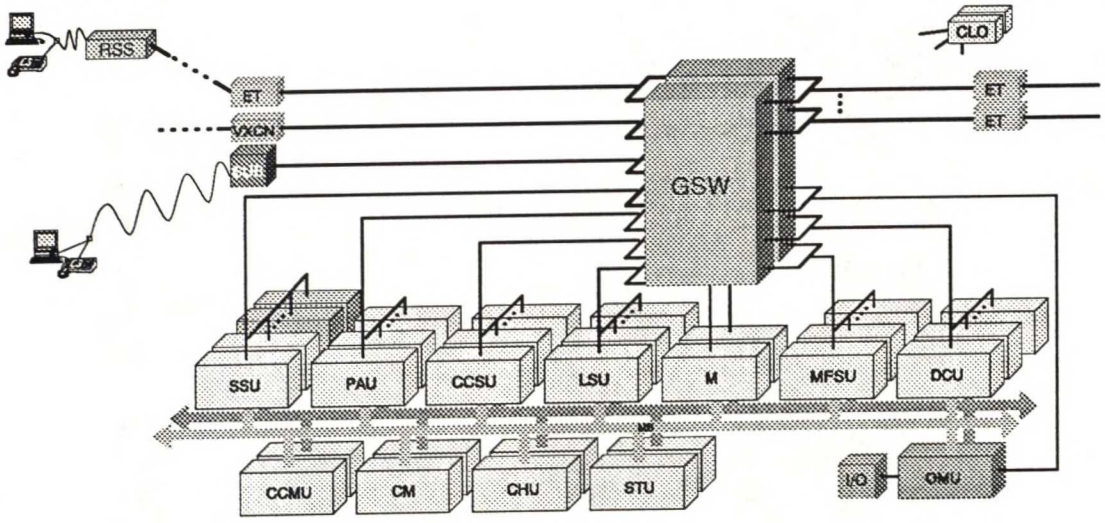
Puhelunohjausosa sisältää puhelinkeskuksen älykkyyden. Siinä sijaitseva ohjelmisto tekee puhelinkeskuksen toiminnassa tarvittavat päätökset. Kytkentäosan tehtävänä on lähinnä kytkeä tilaajajohtoja puhelinohjausosan määräämällä tavalla. Käytönohjausosa on puhelinkeskuksen operaattorille näkyvä osa. Sen avulla voidaan ohjata puhelinkeskuksen toimintaa.



Kuva 1. DX 200 -järjestelmän pääosat

Puhelunohjausosa muodostuu sanomaväylällä toisiinsa yhdistetyistä tietokoneista. Sanomaväylän kautta lähetettyjen sanomien avulla tietokoneet kommunikoivat toistensa kanssa. Jokaisella tietokoneella on oma vastuualueensa. Tietokoneiden määrä ja ohjelmisto ovat riippuvat puhelinkeskuksen tyypistä. Kuvassa 2 puhelunohjausosan tietokoneita kuvaavat laatikot on merkitty vaaleammalla värillä. Niihin kuuluvat mm. SSU (Subscriber Stage Control Unit), CCSU (Common Channel Signalling Unit) ja CM (Central Memory).

Tietokoneiden vikasietoisuutta on parannettu N+1-varmennuksella tai kahdentamalla. N+1-varmennuksessa joukolla tietokoneita on yksi varatietokone. Yhden tietokoneen vikaantuessa varalla oleva tietokone ottaa hoitaakseen tämän tehtävät. Kahdennuksessa jokaisella tietokoneella on oma varayksikkö. Tietokoneisiin voi olla liitettynä kiintolevy-yksikkö tai ne voivat olla levyttömiä. Levy-yksiköt itsessään ovat aina kahdennettuja.



Kuva 2. DX 200 -järjestelmän arkkitehtuuri.

Kytkentäosan muodostaa kytkentäkenttä (GSW, Group Switch), joka on yhteydessä tilaaja- ja yhdysjohtoliitäntään sekä puhelunohjausosaan. Tilaajajohtoliitäntään liittyvät tilaajakohtaiset liittymäjohdot, ja yhdysjohtoliitäntään liittyvät puhelinkeskusten väliset liittymäjohdot.

Käytönohjausosan muodostaa puhelunohjausverkkoon liitetty käytönohjaustietokone (OMU, Operating and Maintenance Unit) oheislaitteineen. Operaattori suorittaa käytönohjaustoiminnot MMI-järjestelmää (Man Machine Interface) käyttäen. Sen

operaattorille näkyvä osa muodostuu joukosta MML-komentoja (Man Machine Language).

Käyttöliittymä on hierarkkinen ja valikkopohjainen. Se muodostuu ylimmällä tasollaan komentoluokista. Kukin komentoluokka jakautuu komentoryhmiksi, jotka muodostuvat yksittäisistä MML-komennoista. Komentoluokilla, komentoryhmillä ja MML-komennoilla on kullakin yksikirjaiminen tunniste. Jokainen MML-komento voidaan tunnistaa yksilöllisesti kunkin tason kirjaintunnisteiden yhdessä muodostamasta kolmikirjaimisesta tunnisteesta. [4]

2.3. OHJELMISTO

2.3.1. Yleistä

DX 200 -järjestelmän ohjelmistoa käytetään puhelinohjaus- ja käytönohjausosan tietokoneissa. Osa ohjelmistosta on yhteistä kaikille tietokoneille ja osa vaihtelee tietokoneen käyttötarkoituksen mukaan.

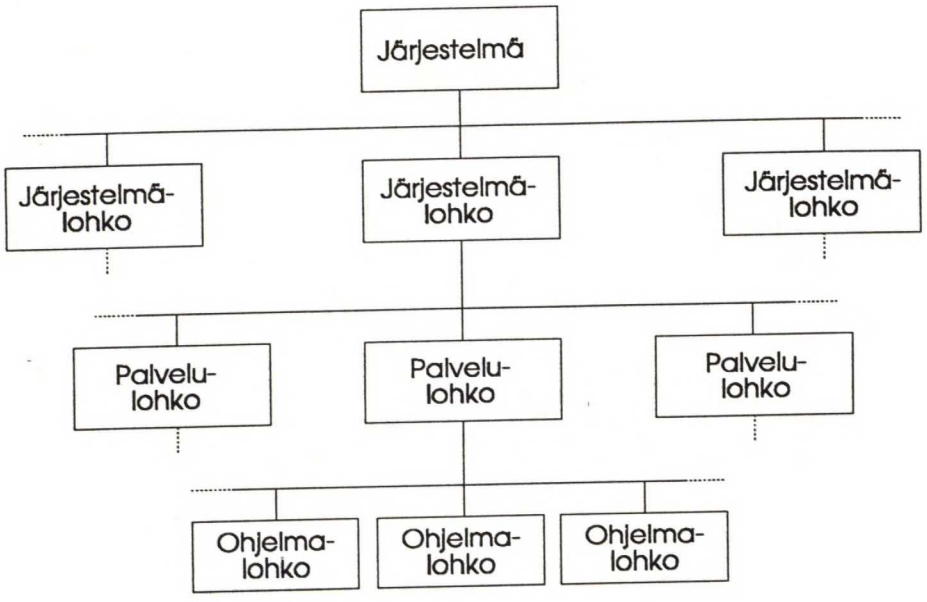
2.3.2. Lohkomalli

Ohjelmisto on mallitettu hierarkkisiin lohkoihin. Ylimpänä hierarkiatasona on järjestelmä kokonaisuudessaan. Se muodostuu joukosta järjestelmälohkoja, jotka vastaavat järjestelmään sisältyvistä toiminnallisista kokonaisuuksista.

Järjestelmälohkot jakautuvat edelleen palvelulohkoihin. Kukin palvelulohko muodostaa järjestelmälohkon sisältä loogisesti yhtenäisen osajoukon.

Hierarkiatason alimpana on ohjelmalohko. Se on yhden selkeästi rajattavan toiminnon toteuttava ohjelma. Palvelulohkot muodostuvat yhdestä tai useammasta ohjelmalohkosta.

Lohkomalli on puurakenne, jossa ohjelmalohko voi kuulua vain yhteen palvelulohkoon ja palvelulohko vain yhteen järjestelmälohkoon. Järjestelmä- ja palvelulohkot ovat hallinnollisia käsitteitä, joilla ohjelmalohkojen tarjoamia palveluita ja niiden edellyttämiä ympäristövaatimuksia voidaan keskitetysti hallita.



Kuva 3. Ohjelmiston lohkomalli.

2.3.3. Ohjelmat

DX 200 -järjestelmä sisältää useita erityyppisiä ohjelmalohkoja, joita käytetään eri käyttötarkoituksiin. Ohjelmalohkoja ovat mm. MML-ohjelmat ja kirjastot. MML-ohjelmat ovat järjestelmän käyttöliittymäohjelmia, jotka huolehtivat käyttäjien komentojen käsittelystä ja niiden toimeenpanosta.

Ohjelmalohkot kommunikoivat toistensa kanssa pääasiallisesti asynkronisesti sanomaväylän kautta välitettävillä sanomilla. Kirjastot ovat ohjelmalohkoja, joiden palveluita muut ohjelmalohkot käyttävät synkronisesti aliohjelmakutsujen tapaan. Niihin voidaan koota järjestelmän toiminnan kannalta keskeisiä palveluita, jotka luonteensa puolesta soveltuvat paremmin synkroniseen käyttöön.

2.3.4. Tiedostot ja tietokannat

Tiedostolla tarkoitetaan DX 200 -järjestelmässä tietokoneen keskusmuistissa sijaitsevaa tiedostoa. Jokaisessa järjestelmän tietokoneessa on tiedostojärjestelmä, joka muodostuu keskusmuistitiedostoista, -hakemistoista ja niiden hallintajärjestelmästä. [5]

Tiedostot ja tietokannat ovat järjestelmän tietovarastoja. Tiedostot soveltuvat yksinkertaisempien ja paikallisempien tietojen tallennukseen. Monimutkaisempien ja laajempien kokonaisuuksien tiedonhallintatarpeisiin soveltuvat paremmin tietokannat.

Tiedostoja hallitseva ohjelmisto on koottu tiedostojärjestelmäkirjastoon. Sen palveluita käyttämällä voivat ohjelmalohkot luoda, lukea ja kirjoittaa tiedostoja.

Tietokanta on loogisesti yhteenkuuluvien tiedostojen kokonaisuus. Sitä hallitsee tietokannan hallintajärjestelmä tiedostojärjestelmän palveluita hyväksi käyttäen. Tietokannan hallintajärjestelmän tehtäviin kuuluvat tietokantatapahtumien suorittamisen lisäksi tietokannan ylläpitoon liittyviä hallinnollisia tehtäviä.

Tiedostosta voi olla olemassa levykopio, jonka ylläpidosta huolehtii erityinen levypäivitysjärjestelmä. Useimmista tietokannoista on olemassa levykopio. Vain ns. työtietokannat ovat kokonaan keskusmuistipohjaisia.

2.3.5. Ympäristömäärittelypaketti

Ympäristömäärittelypaketti on kaikille ohjelmalohkoille näkyvissä oleva tuotekehitysympäristöön muodostettu hakemistorakenne. Sinne talletetaan kaikki useammalle ohjelmalohkolle yhteiset määrittelyt, kuten tietotyyppi-, sanoma-, hälytys-, tiedosto-, virhekoodi- ja palvelumäärittelyt. [6]

Ympäristömäärittelypaketin ansiosta jokainen järjestelmään sisältyvä määrittely tehdään vain yhden kerran ja kaikilla määrittelyä tarvitsevilla ohjelmilla on niistä sama näkemys.

Ympäristömäärittelyt kirjoitetaan TNSDL-kielellä. Ne ovat sellaisenaan käytettävissä TNSDL-kielellä ohjelmoiduille ohjelmille. Assembler-, C- ja PL/M-kielisiä ohjelmia varten generoidaan omat ympäristömäärittelypaketit TNSDL-kielisistä määrittelyistä.

2.3.6. Ohjelmistopaketti

DX 200 -järjestelmässä puhelinkeskuksen ominaisuudet toteuttavat ohjelmat ja tiedostot kootaan ohjelmistopaketiksi. Ohjelmiston lohkomallissa ylimpänä sijaitseva järjestelmä identifioi ohjelmistopakettiin sisältyvät järjestelmä-, palvelu- ja ohjelmalohkot [7].

3. TIETOKANNAT

3.1. YLEISTÄ

DX 200 -järjestelmän tietokantajärjestelmät ja niitä palvelevat levypäivitysjärjestelmät muodostavat monitietokantajärjestelmän, jonka avulla ohjelmalohkot voivat käyttää tietokoneverkon eri pisteissä sijaitsevia ja toisistaan riippumattomia tietokantoja. [8]

Yksittäiset tietokantajärjestelmät muodostuvat tietokannasta ja tietokannan hallintajärjestelmästä. Kukin tietokantajärjestelmä sijaitsee jossakin järjestelmän kahdennetussa tietokoneessa. Tietokannan hallintajärjestelmä huolehtii tapahtumien suorituksen synkronoinnista aktiivisen ja varalla olevan tietokoneen välillä. Lisäksi se huolehtii mm. tietokannan latauksesta sekä vedostuksesta levytietokantaan.

3.2. KÄSITEMALLI

Monitietokantajärjestelmään sisältyvät tietokannat mallitetaan samoilla käsitteillä. Kukin tietokanta muodostuu joukosta objektikokoelmia, -hakemistoja, -relaatioita ja laskennallisia suhteita.

Tietokannan perustan muodostavat tietotyypit. Niiden monimutkaisuus voi vaihdella yksinkertaisista perustietotyyppeistä abstrakteihin tietotyyppeihin. Perustietotyypit sekä erilaiset tietue- ja yhdistelmätyypit ovat käyttökelpoisia ja riittäviä mallinnusvälineitä useissa tapauksissa. Vaativimpien ongelmien mallintamiseen voidaan käyttää abstrakteja tietotyyppejä, joiden määrittely sisältää myös niitä käsittelevät operaattorit.

Objektit ovat tietotyypin omaavia tietokantakäsitteitä. Ne sisältävät tietotyypin lisäksi yksilöllisen objektitunnisteen. Jokainen objekti kuuluu johonkin objektikokoelmaan.

Objektikokoelmat ovat loogisia muistialueita samaa tyyppiä olevien objektien ryhmittelyyn. Niillä on kaksi vaihtoehtoista rakennetta: jono ja taulukko. Jonorakenteesta objekteja voidaan hakea peräkkäishakuna. Taulukkorakenne mahdollistaa lisäksi haun indeksin perusteella. [3]

Objektihakemistot ovat hakurakenteita, joilla objektien hakua objektikokoelmista voidaan tehostaa. Niiden hakuavaimet muodostuvat joukosta objektikokoelman

tietotyypin kenttiä. Objektihakemistoissa voi olla eri objekteille samoja avainarvoja tai ne voidaan määritellä avainarvojen suhteen yksilöiviksi. [3]

Objektikokoelmien väliset yhteydet kuvataan objektirelaatioiden avulla. Objektirelaatiot ovat joukkoja, jonka alkioita ovat objektitunnisteista muodostetut monikot. Ne voivat olla joko kahden tai useamman objektikokoelman välisiä, jolloin niiden sarakemuuttujat jaetaan kahteen ei-tyhjään osajoukkoon, joista toiseen kuuluu vain yksi sarakemuuttuja. Riippuvuudet, jotka muodostetaan näiden osajoukkojen välille, voivat olla siten joko funktionaalisia tai monimääräääviä.

Laskennallinen suhde on objektirelaation eritapaus, jossa kaksi objektikokoelmaa ja yksi yhteyttä kuvaava ominaisuus muodostavat suhteen. Se on muistitilaa säästävä optimoitu ratkaisu erikoistapauksiin. [3]

3.3. TDL-TIETOKANTAKIELI

TDL-tietokantakieli on monitietokantajärjestelmän kuvaamiseen ja käsittelyyn kehitetty määrittely- ja ohjelmointikieli. Se voidaan jakaa neljään loogiseen osajoukkoon: [3]

- loogisen tason tiedonmäärittelykieli
- fyysisen tason tiedonmäärittelykieli
- tiedonkäsittelykieli
- kutsurajapinnan määrittelykieli

Loogisen tason tiedonmäärittelykielellä (DDL, Data Definition Language) kuvataan tietokantakaavio ja tietokantaa käsittelevien tietokantaproseduurien prototyypit. Tietokantakaaviossa kuvataan tietokantaan sisältyvät objektikokoelmat, -hakemistot, -relaatiot sekä laskennalliset suhteet.

Fyysisen tason tiedonmäärittelykielellä (FDL, Physical Data Definition Language) kuvataan tietokannan toteutusmäärittely. Siinä määritellään, miten tiedot on tietokantaan talletettu. Toteutusmäärittely sisältää objektikokoelmien, -hakemistojen, -relaatioiden ja laskennallisten suhteiden talletusrakenteiden kuvaukset. Fyysisen tason tiedonmäärittelykieli tukee erityisesti objektirelaatioiden tehokasta toteutusta. Sen avulla voidaan määritellä kussakin tapauksessa soveltuvin toteutustapa.

Tiedonkäsittelykielellä (DML, Data Manipulation Language) kuvataan tietokantaa käsittelevät proseduurit. Se muistuttaa lohkorakenteisia ohjelmointikieliä, mutta sisältää lisäksi lauseet ja operaattorit objektikokoelmien, -hakemistojen, -relaatioiden ja laskennallisten suhteiden käsittelyyn. Tietokantaproseduureja on kahta lajia: tapahtumia ja pikalukuja. Pikaluvuissa tietokantaan kohdistuvat päivitystoimet on kielletty.

Kutsurajapinnan määrittelykielellä (IDL, Interface Description Language) kuvataan sovelluksen ja tietokannan hallintajärjestelmän välinen rajapinta. Se sisältää sovelluksen näkemän osajoukon tietokantakaaviossa esitellyistä tietokantaproseduureista. Kutsurajapinnan määrittely peittää alleen sovelluksen ja hallintajärjestelmän sanomanvaihdon. Ulkoisesti kutsurajapinta muistuttaa aliohjelmakutsua. [3]

Alla on esitetty esimerkki DDL-kielisestä kuvitteellisen tietokannan tietokantakaaviosta, joka sisältää kaksi objektikokoelmaa, yhden objektihakemiston ja yhden objektirelaation. Kyseessä on sama tietokanta, jota käytetään liitteessä 2 esiteltävässä konversio-ohjelman kuvauksessä lähdetietokantana.

```
DATABASE examdb

    subscribers     ARRAY OF  subscribers_t;
    subsc_services ARRAY OF  subsc_services_t;

    sub_by_sub_no
        DIRECTORY ON subscribers: << subsc_no >>;

    services_of
        RELATION
            sub  IN subscribers;
            serv IN subsc_services;
        WHERE sub <-> serv;

ENDDATABASE examdb;
```

Samalle tietokannalle on laadittu yksinkertaistetulla FDL-kielellä toteutusmäärittely. Selkeyden ja lyhyen esitystavan nimissä toteutusmäärittelystä on poistettu toteutuskohtaisten vakioiden määrittelyt sekä tietokantaproseduurien hyppytaulut. Niillä ei ole merkitystä tietokannan konversio-ohjelmien kannalta.

```
IMPLEMENTATION OF DATABASE examdb
    COLLECTIONS
        subscribers
            fileNumber = 1;
            initValue  = 0;
        END subscribers;
        subsc_services
            fileNumber = 2;
            initValue  = 0;
        END subsc_services;
    DIRECTORIES
        sub_by_sub_no
            fileNumber = 3;
            REPRESENTATION HASH(SUB_HA);
        END sub_by_sub_no;
    RELATIONS
        services_of
            DEPENDENCY ->
                fileNumber = 4;
                REPRESENTATION DENSE;
            DEPENDENCY <-
                fileNumber = 5;
                REPRESENTATION DENSE;
        END services_of;
END examdb;
```

3.4. TIETOKANTATIEDOSTOT

DX 200 -järjestelmän tiukkojen reaaliaikaisuusvaatimusten vuoksi tietokantatiedostot on tallettettu keskusmuistiin. Levypäivitysjärjestelmä ylläpitää levykopioita tietokannan keskusmuistitiedostoista.

Tietokantatapahtumat suoritetaan tietokannan hallintajärjestelmän alaisuudessa. Tapahtumaan sitoutumisen jälkeen hallintajärjestelmä tallettaa tapahtuman tulokset lokipuskuriin, joka välitetään levypäivitysjärjestelmälle. Levypäivitysjärjestelmä kirjoittaa

lokipuskurin välittömästi tietokannan levykopioon liittyvään levylokiin ja aloittaa tapahtumien purkamisen levytiedostoihin.

Tapahtumien purku levylokista levytietokantaan on hitaampaa kuin lokipuskurista keskusmuistitietokantaan. Tästä syystä levytietokanta on hieman jäljessä keskusmuistietokannasta.

4. TIETOKANTAKONVERSIOT

4.1. KONVERSIOTARPEET

4.1.1. Ohjelmistopaketin vaihto

DX 200 -järjestelmän puhelinkeskuksen ominaisuuksia lisätään ja ohjelmistovikoja korjataan vaihtamalla sen ohjelmistopakettia. Usein ohjelmistopakettia vaihdettaessa selvitään joko täysin ilman tai hyvin vähäisillä laitteistomuutoksilla. [9]

Ohjelmistopaketin vaihto on puhelinkeskuksen käytön kannalta kriittinen toimenpide. Sen tietoliikenteelle aiheuttamien haittojen tulisi olla mahdollisimman pieniä. Suuremmissa muutoksissa on varauduttava siihen, että ohjelmistopaketin vaihdon epäonnistuttua voidaan hallitusti palata vanhaan ohjelmistopakettiin.

Uusia ominaisuuksia sisältävän ohjelmistopaketin vaihto edellyttää usein vanhojen tiedostojen ja tietokantojen päivitystä uuden ohjelmiston vaatimaan muotoon. Näitä päivitystoimenpiteitä kutsutaan konversioiksi.

4.1.2. Tiedosto- ja tietokantakonversiot

Konversiot voivat kohdistua joko yksittäisiin tiedostoihin tai tietokantoihin. Tiedostokonversioiden historia on pidempi. Niitä on toteutettu järjestelmän alkuajoista lähtien. Niiden määrä on kuitenkin vähentymässä tietokantojen korvatessa yhä suuremmassa määrin tiedostoja tiedonhallinnan välineinä.

Tietokantakonversiot ovat käytännössä myös tiedostokonversioita, sillä tietokannat muodostuvat joukosta tiedostoja. Tietokantakonversioista puhuttaessa käsitetaso on kuitenkin astetta korkeammalla, sillä niissä konvertoidaan koko tietokannan muodostava tiedostojoukko eheästä tilasta toiseen. Tällöin on tärkeää, että jokainen konvertoinnin tarpeessa oleva tietokantatiedosto tulee asianmukaisesti päivitettyä. Muutoin lopputuloksena saadaan epäeheä tietokanta.

Tietokantakonversioita on edelleen kahta eri tyyppiä, joiden erona on lähtötietojen talletustapa. Tietokantakonversiotyyppejä ovat tiedostoista tietokantaan ja tietokannasta tietokantaan konversiot. Kaikki tietokantakonversiot suoritetaan tietokannan levykopioita hyväksi käyttäen.

Tiedostoista tietokantaan konversioita tarvitaan silloin, kun uudessa ohjelmistopaketissa on perustettu tietokanta korvaamaan vanha yksittäisiin tiedostoihin pohjautunut tiedonhallintajärjestelmä. Ne ovat yleensä kertaluonteisesti siirtymävaiheessa tarvittavia konversioita.

Tietokannasta tietokantaan konversioita tarvitaan tietokantakaavion tai tietotokantatiedostojen muuttuessa ohjelmistopaketista toiseen. Näitä konversiota tarvitaan usein ohjelmistopaketin vaihdon yhteydessä.

Tietokantakonversioiden vaativuus vaihtelee suuresti. Toiset konversiot ovat yksinkertaisia ja ne voidaan suorittaa ajamalla ohjelmistopakettiin kuuluva standardikonversio-ohjelma DEACON. Se käy läpi tietokantatiedostot ja päättelee näiden tarvitsemien konversioiden vaativuuden. Mikäli kaikki konversiotehtävät ovat DEACONin toteutettavissa, se suorittaa konversion tietokannalle.

Yksinkertaisimpiin konversioita vaativiin muutoksiin kuuluvat erilaiset tiedostojen fyysisen talletustavan muutokset. Näihin DEACONin käsittelemiin muutostapauksiin kuuluvat:

- alustusarvon muutos
- tiedostonumeron muutos
- maksimitietuemäärän muutos
- objektirelaation moniulotteisen hajautustaulun dimensiomuutos
- objektirelaation talletusrakenteen muutos

4.2. KONVERSIO-OHJELMAT

Tiedostokonversiot ja vaativammat tietokantakonversiot edellyttävät tapauskohtaisesti laadittuja konversio-ohjelmia. Ne ovat tyypillisesti ohjelmia, jotka ajetaan kerran läpi ja voidaan sen jälkeen poistaa käytöstä.

Konversio-ohjelmia edellyttäviä suurempia tietokantamuutoksia ovat mm.:

- objektikokoelman tietuetyypin muutokset
- objektikokoelman jakautuminen

- objektikokoelmien yhdistyminen
- objektihakemiston hakuvaimen kenttien lisäys tai poisto

Objektikokoelman jakautumisella tarkoitetaan tilannetta, jossa yhdestä objektikokoelmasta muodostetaan kaksi itsenäistä objektikokoelmaa. Objektikokoelmien yhdistymisessä on kyse päinvastaisesta. Siinä kaksi objektikokoelmaa, joiden välillä on objektirelaatio, yhdistetään yhdeksi objektikokoelmaksi.

Konversio-ohjelmien ohjelmointikielinä on käytetty C- ja PL/M-kieliä [10]. Niiden laatimisessa käytetään hyväksi tiedostokonversioiden tukikirjasto FIXLIBiä. Se sisältää rutiineja mm. lähde- ja kohdetiedostojen käsittelyyn. FIXLIB-kirjastoa hyödyntää myös tietokantakonversiokirjasto DEACON, joka sisältää tietokantatiedostojen käsittelyyn erikoistuneita rutiineja. Konversio-ohjelmat voivat käyttää kaikkien FIXLIB- ja DEACON-kirjastojen rutiinien lisäksi MML-ohjelmien MMLLIB-kirjastoa.

Tärkeimmät FIXLIB-kirjaston tarjoamat tiedostokonversiopalvelut ovat: [11]

- tulostus kirjoittimelle ja MML-päätteelle
- levytiedoston luonti
- tiedoston luku levyltä muistiin
- tiedoston osan talletus muistista levytiedostostoon
- tiedoston ominaisuuksien haku
- levytiedoston bittikartan ja tarkistussummien päivitys

DEACON-kirjaston tärkeimmät tietokantakonversioiden palvelut ovat:

- tietokantatiedostojen avaaminen ja sulkeminen
- tietueiden luku ja kirjoitus tietokantatiedostoista
- objektirelaation purku väliaikaistiedostoon
- monikon lisäys objektirelaatioon
- objektirelaation hajautus

Konversio-ohjelmat toteuttavat konversiot levytiedostoja käyttäen. Lähde- ja kohdetiedostot sijaitsevat yleensä eri hakemistoissa.

4.3. KONVERSIO-OHJELMAN YMPÄRISTÖ

4.3.1. Yleistä

Konversio-ohjelmat ovat osa ohjelmistopakettia. Ennen uuden ohjelmistopaketin käyttöönottoa suoritetaan sen mukana tulleet konversio-ohjelmat.

Konversio-ohjelmien ajonaikaisena ympäristönä toimii tiedostokonversioiden hallinnan MML-ohjelma PEXHAN. Se on käyttöliittymäohjelma, jonka komentojen avulla operaattori voi toteuttaa tiedostojen ja tietokantojen konversiot. PEXHAN sisältää komentoja mm. levytiedostojen luontiin, konversio-ohjelmien suorittamiseen, muistitiedostojen kopiointiin levytiedostoihin sekä levytiedostojen bittikarttojen ja tarkistussummien päivitykseen. Lisäksi sen avulla voidaan tarkistaa tiedostojärjestelmän eheys ja vertailla, miten kaksi tiedostojärjestelmää poikkeaa toisistaan. [10]

4.3.2. DX 200 -hakemistorakenne

Ohjelmistopaketin ohjelmat ja tiedostot on jaettu toiminta- tai käyttötapansa mukaan eri hakemistoihin. Juurihakemiston alapuolella on ohjelmistopaketin päähakemisto, jonka alapuolella ovat puolestaan ohjelmistopaketin ohjelmille ja tiedostoille varatut alihakemistot.

Kullekin ohjelmistopaketille on olemassa viisi alihakemistoa. Konversio-ohjelmien kannalta oleellisia hakemistoja näistä ovat CONVPR, LFILES ja MMDIRE. CONVPR on hakemisto konversio-ohjelmille. Käyttäjän antamien komentojen mukaisesti PEXHAN hakee suoritukseen kulloinkin tarvittavan konversio-ohjelman tästä hakemistosta. LFILES on tiedostojen hakemisto, johon talletetetaan sekä yksittäiset tiedostot että tietokantatiedostot. MMDIRE on puolestaan ohjelmistopaketin MML-ohjelmille varattu hakemisto.

4.3.3. Konversio-ohjelman suoritus

Ennen konversio-ohjelman suoritetusta luodaan sen käyttöön työhakemisto, johon kohdetiedostot kirjoitetaan. Työhakemiston luonti tapahtuu levytiedostojen ja -hakemistojen hallinnan MML-ohjelman WINHAN avulla.

Työhakemistoon kopioidaan konversiossa tarvittavat järjestelmätiedostot MAFILE, MEFILE ja DXPFIL. Ne sisältävät tietoja ohjelmistopaketin tiedostoista. MAFILE on ohjelmistopaketin kuvauslista, jolla määritellään kaikki ohjelmistopakettiin kuuluvat tiedostot ja ohjelmat. MEFILE on taulukkotiedosto, johon on koottu kaikkien ohjelmistopakettien tiedostonumerot ja nimet. Sitä käytetään haettaessa tiedostonimeä tiedostonumeron avulla ja päinvastoin. DXPFIL on tiedostojen attribuuttitiedosto, joka sisältää tiedot mm. tiedoston tietuemäärästä ja -koosta.

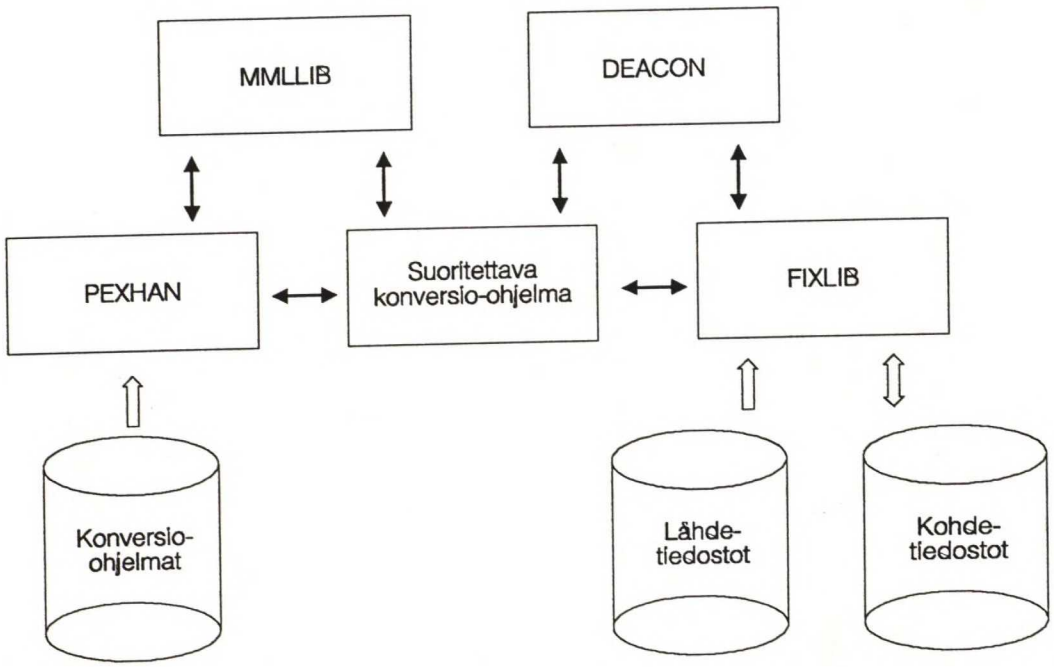
Konversio-ohjelmat ja PEXHAN-käyttöliittymä käyttävät FIXLIB-kirjaston palveluja, jotka puolestaan hyödyntävät tiedostojärjestelmän oletusarvoisia lähde- ja kohdehakemistoja. Käytettävät hakemistot on määriteltävä ennen konversio-ohjelman suoritusta WINHANin komentojen avulla. Lähdehakemistoksi asetetaan vanhan ohjelmistopaketin LFILES-hakemisto ja kohdehakemistoksi edellä luotu työhakemisto.

Näiden alustustoimien jälkeen voidaan aloittaa varsinaisen konversion suoritus. PEXHAN sisältää seuraavat MML-komennot konversioiden suoritukseen: [12]

- DEC Muistitiedoston luonti levylle ja alustus pohja-arvolla.
- DEE Konversio-ohjelman luku levyltä ja suoritus
- DEM Muistitiedoston kopiointi levylle
- DES Muistitiedoston levykopion bittikartan ja tarkissummien päivitys
- DEU Muistitiedostojärjestelmän levykopion eheyden tarkistus
- DEV Kahden muistitiedostojärjestelmän levykopioiden vertailu

Konversio-ohjelman suoritus aloitetaan DEE-komennolla, jolle annetaan parametrina konversio-ohjelman nimi. Konversio-ohjelma on suorituksensa aikana vuorovaikutuksessa ainakin PEXHANin ja FIXLIBin kanssa. Tietokantakonversio-ohjelmat ovat vuorovaikutuksessa lisäksi DEACON-kirjaston kanssa. Mikäli konversio-ohjelma tulostaa suoritustulosteita kirjoittimelle tai MML-päätteelle, se on myös

vuorovaikutuksessa MMLLIBin kanssa. Kaikki konvertoitavien tiedostojen käsittely on keskitetty FIXLIB-kirjastoon.



Kuva 4. Tietokantakonversio-ohjelman suoritusympäristö

5. KUVAUSKIELEN SUUNNITTELU

5.1. YLEISTÄ

Tietokantakonversio-ohjelman laatiminen on vaativa tehtävä. Jokainen konversio-ohjelma on ohjelmoitava toteuttamaan kulloinkin tarvittavat konversiotoimenpiteet. Koska konversio-ohjelmien ohjelmointiin on käytetty C- ja PL/M-kieliä, ohjelmoijan on täytynyt huolehtia varsinaisten konversiotoimenpiteiden ohella mm. rutiininomaisesta tiedostojen käsittelystä. Ohjelmoijan on täytynyt lisäksi perehtyä tarkkaan konversiokirjastojen tarjoamiin palveluihin.

Tietokantakonversio-ohjelmien kuvauskielen kehittäminen nähtiin tarpeelliseksi konversio-ohjelmien laatimistyön helpottamiseksi. Kuvauskielen korkeamman abstraktiotason ansiosta ohjelmoija voisi keskittyä tarvittavien konversiotoimenpiteiden kuvaamiseen. Konversiokirjastojen rajapinnat ja tiedostojen käsittely olisi ohjelmoijilta piilotettu.

5.2. TAVOITTEET

Kuvauskielelle asetettiin useita vaatimuksia. Toiminnallisesta näkökulmasta kuvauskieleltä edellytettiin alustusohjelmien ja tietokannasta tietokantaan konversio-ohjelmien kuvausmahdollisuutta. Kertaluonteiset tiedostoista tietokantaan konversiot jätettiin jatkokehityskohteeksi.

Alustusohjelmilla tarkoitetaan ohjelmia, joita käytetään tietokannan alustamiseen halutuilla alkuarvoilla. Alustus- ja konversio-ohjelmilla on hyvin samankaltaisia piirteitä. Niiden molempien tulee pystyä muuttamaan monin eri tavoin tietokantoihin talletettuja tietoja. Onkin luontevaa, että samalla kuvauskielellä voidaan kuvata myös alustusohjelmia.

Kuvauskielellä tuotetuilta konversio-ohjelmilta edellytetään hyvää suorituskykyä, sillä suuriin tietokantoihin kohdistuvat konversiot ovat usein aikaavieviä toimenpiteitä. Suorituskyvyn merkitystä lisää se, että konversio-ohjelmat suoritetaan yleensä PEXHANin komennoilla interaktiivisesti. Näistä syistä jo kuvauskieltä suunniteltaessa tulee suorituskykynäkökohdat huomioida ja pyrkiä mahdollisimman tehokkaasti toteutettaviin ratkaisuihin.

Ohjelmointitekniseltä näkökannalta kuvauskieleltä vaadittiin määrittelyjen osalta yhteensopivuutta TNSDL-kielen kanssa. Konversio-ohjelmat tarvitsevat poikkeuksellisen luonteensa vuoksi tietoja kahden ohjelmistopaketin ympäristömäärittelyistä. TNSDL-yhteensopivan määrittelysyntaksin ansiosta ympäristömäärittelypaketit ovat suoraan kuvauskielen ymmärtämässä muodossa.

Lähteessä [13] on esitetty joukko ohjelmointikielen arviointiin soveltuvia kriteereitä. Esitetyt kriteerit ovat yleisluonteisia eikä kaikkien ohjelmointikielien tarvitse täyttää niistä kaikkia. Niistä löytyy kuitenkin konversio-ohjelmien kuvauskielen suunnitteluohjeiksi soveltuvia tavoitteita:

Syntaktinen ja semanttinen kuvaus

Selvästi määritelty syntaktinen ja semanttinen kuvaus on olennainen osa ohjelmointikielen suunnittelua. Kaikkien ohjelmointikielten tulee täyttää ainakin tämä kriteeri. Kuvauksen laatimiseen on olemassa hyviä formaaleja kuvaustapoja, kuten BNF (Backus-Naur Form) ja syntaksigraafit. Semantiikan kuvaaminen on formaalisti monimutkaisempaa. Se pitää usein kuvata luonnollisella kielellä kirjoitetulla kuvauksella.

Luotettavuus

Luotettavuudella tarkoitetaan ohjelmointikielen piirteitä, jotka koettavat ehkäistä virheiden syntymistä ja helpottaa niiden löytämistä. Esimerkiksi kommentointitapa voi vaikuttaa ohjelmointikielen luotettavuuteen kahdellakin tapaa. Ensinnäkin selkeät kommentit parantavat ohjelman luettavuutta ja edistävät näin ohjelmien ymmärrettävyyttä. Toiseksi kommentointisyntaksin tulisi olla sellainen, että kommentissa esiintyvät syntaksivirheet havaitaan ohjelmaa käännettäessä. Tässä mielessä vaarallinen kommentointitapa on esimerkiksi ALGOL60-kielessä, jossa kommentti voidaan aloittaa "comment"-sanalla ja päättää puolipisteeseen. Lopettavan puolipisteen unohtuessa kommenttilauseesta tulkitaan virheellisesti myös seuraava ohjelmalause kommenttitekstinä.

Tehokas objektikoodi

Tehokkaan objektikoodin tuottoa voidaan edesauttaa suunnittelemalla ohjelmointikieleen piirteitä, jotka ovat tehokkaasti toteutettavissa käytössä olevalla laitteistolla. Tämä kriteeri tukee konversio-ohjelmien interaktiivisuuden ja tietokantojen suuruuden vuoksi aiemmin edellytettyjä suorityskykyvaatimuksia.

Laiteriippumattomuus

Laiteriippuvuudella tarkoitetaan ohjelmointikielen laitearkkitehtuuriin viittaavia ominaisuuksia, joita ovat mm. sanan pituus ja käytetty merkistö.

Johdonmukaisuus

Johdonmukaisuudella tarkoitetaan yleisesti tunnettujen merkintätapojen käyttöä niiden intuitiivisessä merkityksessä. Esimerkiksi merkin + tulisi tarkoittaa yhteenlaskua kaikille tietotyypeille, joille yhteenlasku on määritelty.

5.3. DEKLARATIIVISET JA PROSEDURAALISET KIELET

Useimmat ohjelmointikielet ovat luonteeltaan proseduraalisia kieliä. Ne sisältävät ennaltamäärätyssä järjestyksessä peräkkäin suoritettavia lauseita. Toiset ohjelmointikielet ovat puolestaan deklaratiivisia kieliä. Niillä ilmaisevat, mitä halutaan tehtäväksi. Päämäärän saavuttamiseksi tarvittaviin toimenpiteisiin deklaratiivinen kieli ei ota kantaa. [14]

Ohjelmointikieli voi sisältää myös automaattisia piirteitä. Niillä tarkoitetaan sitä, että joitakin toimintoja tehdään ilman ohjelmoijan eksplisiittisesti ilmaisemia toiveita. Automaattisten toimintojen suunnittelussa tulee olla varovainen. Kokeneet ohjelmoijat voivat pitää liian pitkälle vietyjä automaattisia toimintoja toimintavapautaan rajoittavina piirteinä. Kokemattomammille ohjelmoijille pitkälle viedyt automaattiset toiminnot ovat sen sijaan käyttöä helpottavia asioita. [14]

Konversio-ohjelmien kuvauskielelle laadittiin kaksi vaihtoehtoa. Toinen vaihtoehdoista oli pääasiassa deklaratiivinen ja toinen proseduraalinen kieli.

5.4. DEKLARATIIVINEN VAIHTOEHTO

5.4.1. Yleistä

Deklaratiivinen kuvauskieli keskittyi tyypillisimpiin konversiotapauksiin. Siinä haettiin kuvaustapaa, jolla yleisimmät konversiotapaukset voitaisiin kuvata selkeästi ja ytimekkäästi. Konversiotoimenpiteet jaettiin tietokantakäsitteiden mukaisesti neljään ryhmään: objektikokoelmien, -hakemistojen, -relaatioiden ja laskennallisten suhteiden konversiotoimenpiteisiin.

5.4.2. Konversiokuvaus

Konversio-ohjelman kuvaus koostuu konversion otsikosta, määrittelyistä, ja konversiotoimenpiteiden kuvauksista. Näistä määrittelyt ja konversiokuvakset ovat vapaaehtoisia.

Määrittelyihin kuuluvat vakion, tietotyypin ja synkronisten palveluiden määrittelyt. Kaikkia määrittelyitä ei tarvitse sisällyttää konversiokuvaukseen, vaan niitä voidaan ottaa mukaan erillisistä määrittelytiedostoista #include-direktiivin avulla.

Konversiokuvauksen otsikossa määritellään lähde- ja kohdetietokannat. Samalla niille määritellään tietokantatunnisteet, joiden avulla voidaan konversiokuvauksessa voidaan yksiselitteisesti ilmaista, kumpaan tietokantaan objektikokoelmat, -hakemistot, -relaatiot ja laskennalliset suhteet sekä tietotyypit liittyvät. Kuvauskieli soveltuu myös alustusohjelmien kuvaamiseen. Alustusohjelmien kuvauksissa ei määritellä lähdetietokantaa.

Konversiotoimenpiteiden kuvauksissa määritellään objektikokoelmille, -hakemistoille, -relaatioille ja laskennallisille suhteille toteuttavia muutoksia. Niille tietokantatiedostoille, joille ei konversiokuvauksissa ole määritelty ja joita ei ole määritelty jätettäväksi konversion ulkopuolelle, tehdään automaattinen konversio.

5.4.3. Konvertoimatta jättäminen

Toisinaan on tarkoituksenmukaista jättää konversiosta pois osa tietokannan objektikokoelmista, -hakemistoista, -relaatioista tai laskennallisista suhteista. Tälläinen tilanne voi esiintyä esimerkiksi testattaessa konversio-ohjelmaa. Kuvauskieleen sisältyy lause, jolla voidaan määritellä osa tietokantatiedostoista jätettäväksi konversion ulkopuolelle.

5.4.4. Objektikokoelmat

Kuvauskielessä on objektikokoelmien konversioille suunniteltu laajin valikoima esimääriteltyjä konversiotehtäviä. Näitä ovat:

- objektikokoelman pohjautuminen toiseen
- objektikokoelman tietuemuutos

- objektikokoelman yhdistyminen
- objektikokoelman jakautuminen

Objektikokoelman pohjautumisella toiseen objektikokoelmaan tarkoitetaan tilannetta, joka syntyy esimerkiksi objektikokoelman nimen muuttuessa. Konversio-ohjelman kuvauksessa on tässä tapauksessa ilmaistava, mikä lähtötietokannan tiedosto vastaa kohdetietokannan tiedostoa. Tiedoston konversiotoimenpiteet toteutetaan näille tiedostoille automaattisesti.

Objektikokoelman tietuemuutos, yhdistyminen ja jakautuminen ovat lauselohkon sisältäviä konversiotehtäviä. Lauselohkon käyttö tuo deklaratiiviseen kieleen rajoitetusti proseduraalisia piirteitä.

Lauselohko voi sisältää sijoituslauseita, ehdollisia haarautumalauseita, kokonaislukumuuttujan ja objektikokoelman toistolauseita sekä ulkopuolisten proseduurien kutsuja. Lauselohkon lausekkeita ja ulkopuolisia proseduurikutsuja hyväksi käyttäen on monimutkaistenkin konversiotehtävien kuvaaminen mahdollista.

Objektikokoelman tietuemuutoksella tarkoitetaan tietuekohtaisesti määriteltyä lauselohkoon perustuvaa konversiota. Siinä kohdeobjektikokoelman toistolauseella haetaan tietueita lähde- ja kohdeobjektikokoelmista ja suoritetaan jokaiselle lauselohkon lauseet.

Objektikokoelman jakautuminen on tietuemuutoksen erityistapaus, jossa yhden lähdeobjektikokoelman pohjalta muodostetaan kaksi kohdeobjektikokoelmaa. Uusien objektikokoelmien välille luodaan lisäksi automaattisesti objektirelaatio, jossa objektien riippuvuus on funktionaalinen.

Objektikokoelmien yhdistäminen on myös tietuemuutoksen variaatio. Siinä muodostetaan kahden lähdeobjektikokoelman perusteella yksi kohdeobjektikokoelma. Lähdeobjektikokoelmien välillä tulee olla määriteltynä objektirelaatio.

Edellisten lisäksi kuvauskieleen sisältyy joukko objektikokoelmiin kohdistuvia automaattisia konversiotoimenpiteitä. Automaattiset konversiot sisältävät tietokantatiedostojen rakenteeseen tai talletustapaan liittyvät muutokset. Tälläisiä muutoksia ovat mm. objektikokoelman tiedoston tietuemäärän, alustusarvon ja tiedostonumeron muutokset. Alustusohjelmissa ei automaattisia toimenpiteitä voida toteuttaa.

5.4.5. Objektihakemistot, -relaatiot ja laskennalliset suhteet

Objektihakemistot, -relaatiot ja laskennalliset suhteet käsitellään viitteellisen kuvauskielen kannalta samankaltaisesti. Jokaiselle niistä voidaan toteuttaa joko automaattisia tai toisiin tietokantatiedostoihin pohjautuvia konversiotoimenpiteitä.

Kaikille yhteisiä automaattisia konversiotoimenpiteitä ovat objektikokoelmien automaattisten konversioiden tapaan tietokantatiedostojen talletustavan muutoksista aiheutuvat konversiot. Niiden lisäksi objektihakemistoihin, -relaatioihin ja laskennallisiin suhteisiin voi kohdistua välillisiä konversiotarpeita objektikokoelmien konvertoinneista.

Objektihakemistoille välillisiä konversiotarpeita aiheuttavat objektikokoelman tietuemuutos, jakautuminen ja yhdistyminen, jotka voivat muuttaa objektikokoelmaan liittyvän objektihakemiston hakuavainten arvoja. Objektirelaatioille ja laskennallisille suhteille välillisiä konversiotarpeita aiheutuu objektikokoelmien yhdistymisessä, jossa aina toisen objektikokoelman objektien indeksijärjestys muuttuu. Tähän objektikokoelmaan liittyvien objektirelaatioiden ja laskennallisiin suhteiden monikoihin tulee päivittää uutta tietuejärjestystä kuvaavat indeksi.

Välillisiä konversioita ei esitellä konversiokuvauksessa. Ne toteutetaan automaattisesti tietokantakaavioiden ja objektikokoelmiin kohdistuvien konversioiden perusteella.

Objektihakemistojen, -relaatioiden ja laskennallisten suhteiden pohjautuminen toisiin tietokantatiedostoihin voi objektikokoelmien tapaan johtua nimen muutoksesta. Konversiotehtävät suoritetaan näille tietokantatiedostoille automaattisesti objektikokoelmien konversioiden tapaan.

5.5. PROSEDURAALINEN VAIHTOEHTO

5.5.1. Yleistä

Proseduraalisen kuvauskielen vaihtoehdossa suurin osa konversiotoimenpiteistä kuvataan ohjelmoiduilla konversiolohkoilla. Niiden lisäksi kuvauskieli sisältää deklaratiivisen tapaan joitakin automaattisesti toteutettavia toimintoja.

Proseduraalisen kuvauskielen syntaksissa päädyttiin suurelta osin DML-kielen kanssa yhtenevään kieliasuun. Samankaltaisen syntaksin käyttö on luontevaa, sillä molemmissa kielissä käytetään keskeisesti samoja tietokantakäsitteitä. Siitä on etua myös

koulutustarvetta arvoitaessa, sillä DML-kieli on entuudestaan tuttu useimmille tietokantakonversio-ohjelmien laatijoille. Samankaltaisen kieliasun ja samojen käsitteiden kautta kynnys uuden kielen käyttöönottoon madaltuu.

5.5.2. Konversiokuvaus

Proseduraalisessa kuvauskielessä konversio-ohjelman kuvaus koostuu deklaratiivisen vaihtoehdon tapaan konversiokuvauksen otsikosta, määrittelyistä ja konversiotoimenpiteiden kuvauksista.

Proseduraalinen kuvauskieli soveltuu deklaratiivisen kuvauskielen tapaan myös alustusohjelmien kuvaukseen. Konversiokuvauksen otsikossa määritellään tällöin tapauksessa vain kohdetietokanta.

Konversiotehtävien kuvaukset käsittävät pääasiassa ohjelmoituja konversioita. Lisäksi niissä voidaan määritellä ne objektikokoelmat, -hakemistot, -relaatiot ja laskennalliset suhteet, jotka halutaan jättää konversion ulkopuolelle tai konvertoida automaattisesti toisesta tietokantatiedostosta.

Automaattiset konversiot, konvertoimatta jättäminen ja toiseen objektikokoelmaan, -hakemistoon, -relaatioon tai laskennalliseen suhteeseen pohjautuminen ovat toteutukseltaan samankaltaisia kuin deklaratiivisessa kuvauskielessä.

5.5.3. Konversiolohko

Proseduraalisen kuvauskielen suurin ero deklaratiiviseen vaihtoehtoon näkyy konversiolohkoissa. Ne ovat ohjelmoituja konversiotoimenpiteitä, joilla kuvataan objektikokoelmille, -hakemistoille, -relaatioille ja laskennallisille suhteille toteutettavat konversiotoimenpiteet.

Kuvauslohkoissa käytettävissä oleva lausekevalikoima on laaja. Kieleen sisältyvät lauseet ovat:

- lausekelause
- objektikokoelman valitsinlause
- objektikokoelman toistolause
- kokonaislukuvälin toistolause

- ehtolause
- hyppylause
- poistumislause

Kaikki lauseet ovat poistumislausetta lukuunottamatta myös DML-kielen lauseita. Lausekelause sisältää sijoituslauseet, synkronisten palveluiden kutsut ja joukon päivityslauseet. Jälkimmäisillä voidaan lisätä objektirelaatioihin ja laskennallisiin suhteisiin monikkoja sekä objektihakemistoon rivejä.

Objektikokoelman valitsinlauseella voidaan valita objektikokoelmasta yksittäinen objekti tai luoda uusi objekti objektikokoelmaan. Objektin valinta voidaan tehdä käyttäen objektin indeksiä tai hakuavaimen arvoa. Objektikokoelman valitsinlause sisältää lisäksi käskylohkon, joka suoritetaan valitulle tai luodulle objektille.

Objektikokoelman toistolauseella haetaan objektikokoelmasta joukko objekteja ja suoritetaan niille lauseessa määritelty käskylohko. Haettavien objektien määrää voidaan rajoittaa käyttämällä valintakriteerejä.

Kokonaislukuvälin toistolauseella voidaan laatia silmukoita. Ehtolausetta käytetään konversiolohkossa haarautumiseen. Hyppylaiseella voidaan toistolauseista poistua ennen toiston loppuehdon täyttymistä. Poistumislauseella voidaan konversio-ohjelman suoritus keskeyttää ennenaikaisesti.

5.6. KUVAUSKIELEN VALINTA

Konversio-ohjelmien kuvauskieleksi valittiin proseduraalinen vaihtoehto. Molemmat kuvauskielen vaihtoehdot täyttivät kuvauskielelle asetetut tavoitteet. Eroja on sen sijaan kuvauskielet toteuttavien esikääntäjien laatimisen vaativuudessa ja kuvauskielten soveltuvuudessa monimutkaisimpien konversiotoimenpiteiden kuvaamiseen.

Deklaratiivinen kuvaustapa sisältää muutamia hankalasti toteutettavia piirteitä, jotka aiheutuvat objektikokoelmien konversioiden välillisestä vaikutuksesta niihin liittyviin -hakemistoihin, -relaatioihin ja laskennallisiin suhteisiin. Välillisten konversiotarpeiden selvittäminen vaatisi kuvauskielen toteuttavalta esikääntäjältä tietokantakaavion tarkkaa analyysia.

Proseduraalisessa kuvauskielen vaihtoehdossa tälläisten objektihakemistojen, -relaatioiden ja laskennallisten suhteiden päivityksistä huolehtii konversiokuvauksen laatija. Kuvauskielen joukonpäivityslauseilla näiden päivitysten ohjelmointi on kuitenkin helppoa ja luontevaa. Objektikokoelman ja siihen liittyvien objektihakemistojen, -relaatioiden ja laskennallisten suhteiden konversiotoimenpiteistä voidaan laatia yksi yhteinen konversiolohko, jossa kaikki näihin liittyvät tiedostot konvertoidaan kerralla.

Proseduraalisen kuvauskielen valintaan toteutettavaksi kuvauskieleksi vaikuttivat edellä mainitut deklaratiivisen vaihtoehdon vaikeasti toteutettavat ja konversion selkeyttä haittaavat piirteet. Esikääntäjän laatiminen proseduraaliselle kuvauskielelle on yksinkertaisempaa kuin deklaratiiviselle. Proseduraalisen kuvauskielen syntaksin samankaltaisuus DML:n kanssa on myös tärkeä tekijä kuvauskieltä käyttöönotettaessa. Kuvauskielen käyttäjille suunnattavan koulutuksen tarve on selvästi vähäisempää kuin kokonaan uuden syntaksin tarjoavan deklaratiivisen kuvauskielen tapauksessa. Kolmas merkittävä ero kuvauskielten välillä johtuu niiden erilaisista lähtökohdista. Deklaratiivisessa kuvauskielessä pyritään etukäteen arvoimaan tarvittavat konversiot ja laatimaan kuvaustapa niitä varten. Erikoisempien ja harvinaisempien konversiotoimenpiteiden kuvaamiseen se ei tarjoa välineitä. Ne onkin ohjelmoitava C- tai PL/M-kielisinä aliohjelmina. Proseduraalinen kuvauskieli on täysimittainen ohjelmointikieli, joka sisältää monipuoliset lauserakenteet hyvin monimutkaistenkin konversiotoimenpiteiden kuvaamiseen. C- tai PL/M-kielisille aliohjelmille ei ole nähtävissä samanlaista käyttötarvetta kuin deklaratiivisen kuvauskielen tapauksessa.

6. KUVAUSKIELEN SYNTAKSI

6.1. ESITYSTAPA

Kuvauskielen syntaksin esitystapa perustuu vuonna 1963 julkaistussa ALGOL60 - raportissa esiteltyyn notaatioon, jota kutsutaan kehittäjiensä mukaan BNF-esitystavaksi. Siinä syntaksi kuvataan metakielellä, joka muodostuu produktioista eli tuottosäännöistä. Produktiot voivat sisältää välikkeitä, päätesymboleja sekä erotinmerkkejä. Välikkeet ovat metakielen symboleja, jotka edustavat niille asetettua metalausekkeiden luokkaa. Päätesymbolit ovat kuvattavaan kieleen sellaisenaan kuuluvia merkkijonoja. [15]

Produktio muodostuu vasemmalla puolella olevasta välikkeestä, erotinmerkistä ::= ja oikeanpuoleisesta metalausekkeesta. Vasemmanpuoleisen välikkeen merkitykseksi asetetaan oikealla puolella esiintyvä metalauseke.

Metalauseke voi muodostua välikkeistä ja päätesymboleista. Myös tyhjä lauseke on sallittu. Metalauseke voi sisältää myös vaihtoehtoisia lausekkeita, jotka on erotettu erotinmerkillä |.

Esimerkki BNF-esitystavan mukaisesta produktiossa, jossa tunnisteista muodostuva lista. Päätesymbolit on esitetty vahvennettuina ja välikkeet kursiivilla:

identifierList ::=
    identifier [{ , *identifier* }]

BNF-esitystapaan on toteutettu useita laajennuksia. Mm. lähteissä [13] ja [16] on esitelty laajennus, joka mahdollistaa toiston ja vapaaehtoisuuden esittämisen tiiviimmässä muodossa. Konversio-ohjelmien kuvauskielen esitystapa on vastaavasti laajennettu BNF-esitystapa, jossa kaarisuluilla ilmaistaan toisto ja hakasuluilla vapaaehtoisuus. Toistolla tarkoitetaan kaarisulkujen sisällä esiintyvän lausekkeen toistoa yksi tai useampikertaisesti. Vapaaehtoisuudella tarkoitetaan puolestaan hakasulkujen sisällä esiintyvän lausekkeen mahdollisuutta esiintyä kerran tai ei kertaakaan.

Metasymboli	Merkitys
identifier	Välike
IF	Päätesymboli
::=	Erotinmerkki, jota käytetään erottamaan produktion vasemmalla puolella oleva välike oikeanpuoleisesta metalausekkeesta
|	Erotinmerkki, jota käytetään erottamaan produktion vaihtoehtoisia metalausekkeita
{ }	Toistosymboli, jota käytetään esittämään kaarisulkujen sisällä olevan metalausekkeen toistomahdollisuutta yhden kerran tai useammin
[ ]	Valinnaisuussymboli, jota käytetään esittämään hakasulkujen sisällä olevan metalausekkeen esiintymismahdollisuutta yhden kerran tai ei kertaakaan

Taulukko 1. Kuvauskielen esittämiseen käytetyt metasymbolit

6.2. LEKSIKAALISET SÄÄNNÖT

Kuvauskielen leksikaaliset säännöt noudattavat TDL- ja TNSDL-kielien leksikaalisia sääntöjä. Leksikaalisiin sääntöihin kuuluvat kommentit, tunnisteet, numeeriset vakiot, merkkijonovakiot ja varatut sanat.

6.2.1. Kommentti

Kuvauskieli tukee kahta kommenttilajia. Toisessa kommentti alkaa /* merkeillä ja päättyy loppumerkkeihin */. Kommenttimerkkien välillä voi esiintyä vapaamuotoista tekstiä poislukien merkkiyhdistelmän */.

Toista kommentointitapaa voidaan käyttää kuvauskielessä aina puolipistettä edeltämässä. Tässä kommentointitavassa tekstin esitystapa on muodollisempaa. Tätä kommentointitapaa kuvaa seuraava produktio:

end ::=

[COMMENT stringConstant] ;

6.2.2. Tunniste

Tunnisteet muodostuvat kirjaimista, alleviivausmerkeistä ja numeroista. Tunniste alkaa aina kirjaimella tai alleviivausmerkillä. Isot ja pienet kirjaimet tulkitaan tunnisteissa eri merkeiksi.

Tunniste ei saa olla kuvauskielen tai C-kielen varattu sana. Lisäksi käytössä olevat C-kielen käännöstyökalut rajoittavat tietotyypin operaattorin tai sisäisen vakion nimen pituuden ja tietotyypin nimen pituuden summan enintään 30 merkkiin. Muut tunnisteet saavat olla enintään 31 merkkiä pitkiä.

6.2.3. Numeerinen vakio

Numeerisia kokonaislukuvakioita voidaan kielessä esittää kymmen-, heksadesimaali- ja binäärijärjestelmässä. Heksadesimaalivakioiden edessä käytetään etuliitettä 0x (tai 0X) ja binäärivakioiden edessä etuliitettä 0b (tai 0B). Kymmenjärjestelmän luvuilla ei käytetä etuliitteitä. Heksadesimaalijärjestelmässä lukuja 10_{10}:stä 16_{10}:een merkitään aakkosilla A:sta F:ään tai vastaavilla pienaakkosilla. Binäärijärjestelmässä ovat luonnollisesti sallittuja vain luvut 0 ja 1.

Reaalilukuvakiot esitetään aina kymmenjärjestelmässä. Reaalilukuvakiot muodostuvat kokonaislukuosasta, desimaaliosan erottavasta pisteestä, desimaaliosasta ja exponenttiosasta. Desimaaliosa ja exponenttiosa ovat vapaaehtoisia. Exponenttiosassa etuliitteenä on E (tai e).

Tunnisteet ja numeeriset vakiot on esitetty säännöllisten lausekkeiden avulla. Käytetyssä merkintätavassa [a-z] tarkoittaa yhtä merkkiä arvoväliltä a:sta z:aan päätepisteet mukaanlukien. Merkillä + ilmastaan toistoa yhden tai useamman kerran, ja merkillä * toistoa nolla tai useamman kerran. Merkkillä ? ilmaistaan valinnaisuutta, ja merkillä . mitä tahansa merkkiä. Merkeillä ' ympäröidyt merkit esiintyvät lausekkeissa sellaisenaan ilman edellä mainittuja erikoismerkityksiä. Lisäksi merkkejä ( ) voidaan käyttää lausekkeiden ryhmittelyyn.

tunniste	[_a-zA-Z][_a-zA-Z0-9]*
desimaalivakio	[0-9]+
heksadesimaalivakio	0[xX][0-9a-fA-F]+
binäärivakio	0[bB][01]+
reaalivakio	[0-9]+'.'[0-9]*([Ee][+—]?[0-9]+)?

Taulukko 2. Tunnisteiden ja numeeristen vakioiden säännölliset lausekkeet

6.2.4. Merkkijonovakio

Merkkijonovakioille on kaksi esitystapaa. Normaalissa esitystavassa merkkijonovakio alkaa heittomerkillä ja päättyy heittomerkkiin. Jos merkkijonovakiossa halutaan esittää heittomerkki, on se esitettävä kahdennettuna. Esimerkiksi sana vaa´ankieli on kirjoitettava merkkijonovakiona muodossa

´vaa´´ankieli´

Toinen merkkijonojen esitystapa soveltuu paremmin teksteihin, jotka sisältävät runsaasti heittomerkkejä. Tässä esitystavassa merkkijonovakio alkaa kahdella käänteisellä heittomerkillä ja loppuu myös samanlaiseen merkkipariin. Merkkijonovakiossa esiintyviä heittomerkkejä ei tässä esitystavassa tarvitse kahdentaa. Edellisen esimerkin merkkijonovakio kirjoitetaan tässä esitystavassa seuraavasti:

``vaa´ankieli``

Merkkijonovakiot voidaan aina jakaa useammille riveille. Peräkkäiset, valkoisilla merkeillä[1] erotetut merkkijonovakiot yhdistetään yhdeksi merkkijonoksi, joiden yhdistymiskohtaan lisätään rivinvaihtomerkki.

Esimerkiksi seuraavat kaksi merkkijonoa tulkitaan yhdeksi merkkijonoksi:

´Esimerkki pitkästä merkkijonosta,´
´joka on jaettu useammalle riville.´

[1]väliluonti-, sarkain-, rivinvaihto- tai sivunvaihtomerkeillä

6.2.5. Erikoismerkki

Kuvauskieleen sisältyy joukko erikoismerkkejä, joita käytetään erottimina ja operaattoreina.

special ::=

```
@ | $ | # | -> | => | << | >> |
<- | ->> | <<- | <-> | <<-> | <->> | <<->> |
: | ; | , | . | ( | ) |
= | /= | > | >= | < | <= |
+ | - | * | / |
```

6.2.6. Varattu sana

Varattuja sanoja ovat kaikki kuvauskielessa esiintyvät sanat, joilla on syntaksissa erikoismerkitys. Niitä ei saa käyttää muissa yhteyksissä kuin jäljempänä määriteltävissä erityistehtävissä.

Varatut sanat on esitetty suuraakkosin, mutta voidaan ne yhtäpitävästi esittää joko pienillä tai suurilla aakkosilla kirjoitettuna.

keyword ::=

A | ABSTRACT | AND | ARRAY | AT | BITS | BITSTRUCT
| BRANCH | BREAK | BY | COLLECTIONS | COMMENT
| COMPUTATIONAL | COMPUTATIONALS | CONSTANT
| CONVERSION | CONVERT | DATABASE | DENSE
| DEPENDENCY | DIRECTORIES | DIRECTORY | DO
| EACH | ELSE | ELSEIF | END | ENDBITSTRUCT
| ENDDATABASE | ENDENUM | ENDLIBRARY | ENDSTRUCT
| ENDTYPE | ENDUNION | ENUM | EXIT | FAR | FASTREADS
| FILE | FOR | FROM | GRID | HASH | IF | IMPLEMENTATION | IN
| INDEXES | IS | LIBRARY | MOD | NEAR | NEW | NOT | OF | ON
| OPERATORS | OR | OTHERWISE | OUT | PACKED | POINTER
| PROCEDURE | PROCESS | RELATION | RELATIONS
| RELATIONSHIP | REPRESENTATION | SEQUENCE
| SERVICES | SKIP | SOME | SPARSE | STRUCT | SUBSET
| SUCH | SYNC | TABLE | THAT | THE | THEN | TO
| TRANSACTION | TRANSACTIONS | TYPE | UNION | UNIQUE
| USING | VIEWED | WEAK | WHERE | WITH

6.3. MÄÄRITTELYT

6.3.1. Yleistä

Määrittelyihin sisältyvät vakioiden, tietotyyppien ja synkronisten palveluiden määrittelyt. Ne ovat pieniä poikkeuksia lukuunottamatta samanlaisia kuin TNSDL-kielessä esiintyvät määrittelyt.

6.3.2. Vakiot

Vakiomäärittelyllä voidaan nimetä vakiolauseke. Nimettyä vakiota voidaan käyttää lausekkeen sijasta kaikkialla. Vakiomäärittelyitä on näkyvyysalueeltaan kahden tyyppisiä: tietotyypin sisäisiä ja globaaleja. Tietotyypin sisäiset vakiot käsitellään tietotyypin määrittelyn yhteydessä. Globaalit vakiot määrittelevät seuraavat produktiot:

constantDefinition ::=
    CONSTANT { *constantDeclaration* }

constantDeclaration ::=
    constantIdentifier = *constantExpression end*

6.3.3. Tietotyypit

Käyttäjän määrittelemät tietotyypit voidaan jakaa kahteen ryhmään: abstrakteihin ja normaaleihin. Abstraktien tietotyyppien toteutus on piilotettu käyttäjiltä. Niitä voidaan käsitellä ainoastaan tyypille määriteltyjen operattorien kautta.

Tyypinmäärittelyn yhteydessä voidaan nimetä tyypinsisäisiä vakioita. Määrittelylauseen ulkopuolella näihin on viitattava etuliitteellä, joka muodostuu tyypin nimestä ja alaviivasta.

Tyypinmäärittelystä on jätetty pois TNSDL-kieleen määritelty, mutta vielä toteuttamaton tyyppigeneraattorin käsite. Mikäli piirre toteutetaan TNSDL-kieleen, voidaan se ottaa mukaan myös kuvauskieleen.

typeDefinition ::=
    [**ABSTRACT**] **TYPE** *typeIdentifier*
        [*constantDefinition*]
        REPRESENTATION *typeDeclaration end*
        [**OPERATORS** { *operatorDefinition* }]
    ENDTYPE [*typeIdentifier*] *end*

Tietotyypin esitystavaksi voidaan asettaa esimääritelty tai aiemmin määritelty tietotyyppi. Muita mahdollisia tietotyypin arvoja ovat arvoalueeltaan rajatut kokonaislukutyypit, luetellut tyypit sekä bitti-, taulukko-, tietue-, yhdistelmä- ja osoitintyypit.

typeDeclaration ::=
    typeIdentifier
    | *integerSubrangeDeclaration*
    | *enumarationDeclaration*
    | *bitsDeclaration*
    | *arrayDeclaration*
    | *structDeclaration*
    | *bitStructDeclaration*
    | *unionDeclaration*
    | *pointerDeclaration*

Kokonaislukuvälin alkuarvon tulee olla pienempi kuin loppuarvo. Välin päätepisteet ovat mukana sallitussa arvoissa.

integerSubrangeDeclaration ::=
    (*constantExpression* **:** *constantExpression* **)**

Lueteltu tyyppi muodostuu joukosta vakioita. Vakiot numeroidaan oletusarvoisesti 0:sta eteenpäin. Vakioille voidaan määritellä myös muita arvoja, jolloin oletusarvoinen numerointi jatkuu viimeisestä määritellystä arvosta. Kaikkien vakioiden arvojen on oltava yksilöllisiä.

Luetellun tyypin vakioihin viitataan tunnuksella, joka muodostuu luetellun tyypin nimestä, alaviivasta sekä varsinaisesta vakion nimestä.

enumerationDeclaration ::=
    ENUM

enumConstantDeclaration [{ , *enumConstantDeclaration* }]
ENDENUM

enumConstantDeclaration ::=
constantIdentifier [= *constantDeclaration*]

Bittityyppi on etumerkitön kokonaisluku, joka muodostuu 1-32 bitistä.

bitsDeclaration ::=
BITS (*constantExpression*)

Taulukkotyyppi muodostuu joukosta samanlaisia perustyyppejä. Taulukko indeksoidaan aina 0:sta lähtien.

arrayDeclaration ::=
ARRAY { *dimension* } **OF** *typeDeclaration*

dimension ::=
(*constantExpression*)

Tietuetyyppi muodostuu joukosta kenttiä. Tietue voidaan pakata, jos kaikki sen sisältämät kentät ovat tietotyypiltään ovat kokonaislukuja (tai kokonaislukuvälejä), lueteltuja tyyppejä, bool- tai character-tyyppejä [2].

structDeclaration ::=
[**PACKED**] **STRUCT** { *fieldDeclaration* } **ENDSTRUCT**

fieldDeclaration ::=
fieldIdentifier typeDeclaration end

Bittitietuetyyppi on tietuetyypin erityistapaus. Siinä kaikkien kenttien tyyppinä on etumerkitön kokonaisluku, jonka pituus ilmaistaan bittimääränä. Bittitietuetyyppi on vanha TNSDL-kielessä käytetty tietorakenne, joka on mukana yhteensopivuussyistä. Uudet bittitietueet tulisi määritellä pakattuina tietueina bittityyppien avulla. [17]

bitStructDeclaration ::=
BITSTRUCT { *bitFieldDeclaration* } **ENDBITSTRUCT**

bitFieldDeclaration ::=
fieldIdentifier constantExpression end

Yhdistelmätyypin määrittely muistuttaa tietueen määrittelyä. Yhdistelmässä kentät sijaitsevat kuitenkin päällekkäin.

unionDeclaration ::=
UNION { *fieldDeclaration* } **ENDUNION**

Osoitintyypin määrittelyssä on ilmaistava osoittimen kantatyyppi. Osoitintyyppiä määriteltäessä sen kantatyyppi saa olla vielä määrittelemätön.

pointerDeclaration ::=
POINTER [**NEAR** | **FAR**] (*typeIdentifier*)

6.3.4. Synkroniset palvelut

Synkronisten palveluiden esittelyjä tarvitaan kaikista niistä synkronisista palveluista, joita alustus- tai konversio-ohjelma kutsuu.

syncServiceDefinition ::=
SERVICES SYNC { *syncServiceDeclaration* }

syncServiceDeclaration ::=
operatorDefinition
| *viewedDefinition*
| *libraryDefinition*

Operaattorin määrittelyssä voidaan esitellä funktion nimi, parametrit, paluuarvo sekä kutsutapa.

operatorDefinition ::=
[**PROCEDURE**] *operatorIdentifier* ([*formalParameterList*])
-> [*typeIdentifier*] [, *implementationDefinition*] *end*

formalParameterList ::=
formalParameter [{ , *formalParameter* }]

formalParameter ::=
passingMethod parameterIdentifier typeIdentifier

Funktion parametrin voidaan välittää joko arvoparametreina (IN) tai muuttujaparametreina (IN/OUT). Välitystapa voidaan ilmaista myös C-kielen tapaan, kuten vanhemmissa TNSDL-kielissä palvelumäärittelyissä on käytetty. [17]

passingMethod ::=
IN
| **IN / OUT** [**NEAR** | **FAR**]

implementationDefinition ::=
[**NEAR** | **FAR**] [**=>** | **<=**]

Synkronisiin palveluihin kuuluvat myös VIEWED-palvelut. Niiden esittely ja käyttö muistuttavat muuttujia. VIEWED-palvelun ero muuttujiin on siinä, että sovellusohjelma ei voi muuttaa VIEWED-muuttujan arvoa.

viewedDefinition ::=
VIEWED [**NEAR** | **FAR**] *viewedDeclaration* [{ , *viewedDeclaration* }] *end*

viewedDeclaration ::=
viewedIdentifierList typeDeclaration

Synkronisia palveluita voidaan koota myös kirjastoiksi

libraryDefinition ::=
LIBRARY *libraryIdentifier end*
{ *operatorDefinition* | *viewedDefinition* }
ENDLIBRARY *libraryIdentifier end*

6.4. KÄÄNNÖSDIREKTIIVIT

Kuvauskieli sisältää joukon käännösdirektiivejä. Näillä direktiiveillä ohjataan käännöksen kulkua. Kaikki direktiivit alkavat merkillä # ja ne kirjoitetaan omalla rivillään.

6.4.1. Tiedoston sijoitus

Direktiivillä #include määritellään tiedosto, joka sijoitetaan kokonaisuudessaan direktiivin paikalle kuvaustiedostossa. Sijoitettava tiedosto voi edelleen sisältää #include-direktiivejä. Sisäkkäisiä sijoituksia tuetaan ANSI-C -standardin edellyttämään 8 tasoon saakka [18]. TNSDL- ja TDL-kielet sisältävät myös #include-direktiivin.

Hakemistomäärityksissä voidaan käyttää esimääriteltyjä loogisia nimiä ympäristömäärittelypaketin hakemistoihin viittaamiseen. Hakemistotasot erotetaan

jakoviivalla. Esimerkiksi järjestelmätason määrittelytiedosto voidaan sijoittaa konversiokuvaukseen direktiivillä:

```
#include "s_dty:/dxsystgx.sdt"
```

6.4.2. Ehdollinen kääntäminen

Ehdollinen kääntäminen on toteutettu joukolla direktiivejä. Kuten #include-direktiivi edellä nämäkään direktiivit eivät näy kuvauskielen jäsentäjälle, vaan niillä ainoastaan ohjataan kääntäjän toimintaa selaustasolla. Ehdollisen kääntämisen direktiivit ovat samat kuin TDL-kieleen toteutetut. Ne muodostuvat osajoukosta C-kielen ehdollisen kääntämisen direktiiveistä [3].

Symbolien määrittely

Ehdollisen kääntäminen perustuu määriteltäviin symboleihin. Näiden avulla voidaan ilmaista esimerkiksi käyttöympäristöön liittyviä asioita.

Symboleille pätevät samat säännöt kuin tunnisteille. Ne voivat sisältää alleviivausmerkkejä, kirjaimia ja numeroita. Ensimmäisen merkin tulee kuitenkin olla alleviivausmerkki tai kirjain. Symbolit voivat sisältää sekä isoja että pieniä kirjaimia. Isot ja pienet kirjaimet tulkitaan eri merkeiksi.

Direktiivillä #define määritellään symboli, ja direktiivillä #undef poistetaan symbolin määrittely. Esimerkki symbolien määrittelyistä:

```
#define BLACK_PACKET
#undef DEBUG
```

Symbolien tarkistus

Ehdollinen kääntäminen toteutetaan symbolien tarkistusdirektiiveillä #ifdef ja #ifndef. Ensimmäisellä direktiivillä tarkistetaan, onko symboli määritelty, ja jälkimmäisellä, onko symboli määrittelemätön. Tarkistus voi sisältää yhden #else-direktiivin, ja se päättyy aina #endif-direktiiviin. Tarkistusdirektiivejä voidaan käyttää vapaasti sisäkkäin.

Jos tarkistus on tosi, käännetään kuvaustiedoston rivejä normaaliin tapaan, kunnes kohdataan #else tai #endif. Mikäli #else kohdataan, jätetään #else:n ja #endif:n väliset rivit kääntämättä.

Mikäli tarkistus on epätosi, jätetään kuvaustiedoston rivit kääntämättä siihen saakka, että kohdataan #else tai #endif. Tässä tapauksessa mahdollisen #else:n ja #endif:n väliset rivit käännetään.

Esimerkki symbolien tarkistusdirektiiveistä:

```
#ifdef BLACK_PACKET
/* Mustan ohjelmistopaketin lauseita */
#else
/* Muiden kuin mustan ohjelmistopaketin lauseita */
#endif
```

6.5. KONVERSIOKUVAUS

Tietokantakonversio-ohjelma kuvataan määrittelyjä lukuunottamatta yhdessä konversiokuvaustiedostossa. Konversiokuvaus muodostuu otsikosta, määrittelyistä ja konversiotoimenpiteistä. Määrittelyosuus voi sisältää vakioden, tietiotyyppien ja synkronisten palveluiden määrittelyjä.

conversionProgramDefinition ::=
CONVERSION *conversionIdentifier*
[**FROM** *sourceIdentifier databaseAttributes*]
TO *destinationIdentifier databaseAttributes*
[{ *definition* }]
[{ *conversionDefinifion* }]
END [*conversionIdentifier*] *end*

Konversiokuvauksen otsikossa määritellään lähde- ja kohdetietokannat sekä kiinnitetään näihin tietokantatunnisteet. Alustusohjelmien kuvauksissa määritellään vain kohdetietokanta.

databaseAttributes ::=
DATABASE : *databaseIdentifier end*
FILE : *fileIdentifier end*
DIRECTORIES : *directoryIdentifierList end*

Tietokannan määrittely muodostuu joukosta tietokanta-attribuutteja. Näillä attribuuteilla määritellään tietokannan nimi, toteutusmäärittelytiedoston nimi ja määrittelytiedostojen

hakemistopolut. Toteutusmäärittelytiedosto sisältää #include-direktiivin, jonka avulla myös tietokantakaavio luetaan.

Toteutusmäärittelytiedosto analysoidaan ja sen perusteella voidaan päätellä osa automaattista konversioista. Sitä samoin kuin tietokantakaaviota käytetään lisäksi konversiotoimenpiteiden laillisuuden käännösaikaisissa tarkistuksissa.

Alla on esimerkki yksinkertaisesta konversio-ohjelman kuvauksesta, joka ei sisällä ainuttakaan konversiotoimenpidettä. Liittessä 2 on esitetty laajempi esimerkki konversio-ohjelman kuvauksesta.

```
CONVERSION simple
        FROM source
                DATABASE: example;
                FILE: examplegx.fdl;
                DIRECTORIES: /source/dty;
        TO destination
                DATABASE: example;
                FILE: examplegx.fdl;
                DIRECTORIES: /destin/dty;
END simple;
```

Konversiotoimenpiteiden kuvauksissa määritellään objektikokoelmille, -hakemistoille, -relaatioille ja laskennallisille suhteille tehtävät konversiotoimet. Kuvauskielen syntaksiin sisältyvät konversiotoimenpiteet ovat:

- konversio toisen tietokantatiedoston pohjalta
- ohjelmoitu konversiolohko

Näiden lisäksi tehdään konversiotoimenpiteenä automaattinen konversio niille tietokantatiedostoille, joille ei konversiokuvauksessa ole toimenpiteitä määritelty ja joita ei ole määritelty jätettäväksi konversion ulkopuolelle.

Alustusohjelmien kuvauksissa kaikki toimenpiteet on ohjelmoitava konversiolohkoina.

conversionDefinition ::=
excludeDefinition
| *basedOnDefinition*
| *conversionBlockDefinition*

6.6. KONVERTOIMATTA JÄTTÄMINEN

Toisinaan voi olla tarkoituksenmukaista jättää konversiosta pois osa tietokannan objektikokoelmista, -hakemistoista, -relaatioista tai laskennallisista suhteista. Tälläinen tilanne voi esiintyä esimerkiksi testattaessa konversio-ohjelmaa. Alustusohjelmien kuvauksessa tätä lausetta ei käytetä.

excludeDefinition ::=
SKIP *excludeIdentifierList* **END** *end*

Esimerkki, jossa objektikokoelma centrexes ja objektirelaatio cx_abbreviates_of on jätetty konversion ulkopuolelle.

```
SKIP
        centrexes,
        cx_abbreviates_of
END;
```

6.7. AUTOMAATTISESTI TEHTÄVÄT KONVERSIOT

Tietokantatiedoston fyysisen talletustavan muuttuessa se täytyy konvertoida uutta talletustapaa vastaavaan muotoon. Talletustapa voi muuttua tiedostojärjestelmässä toteutetun parametrimuutoksen tai tietokannan toteutusmäärittelytiedostoon tehdyn muutoksen seurauksena.

Talletustavan muutos ei vaikuta tietokannan loogiselle tasolle. Tästä syystä talletustavan muutoksesta aiheutuvat konversiot soveltuvat hyvin automaattisesti toteutettaviksi.

Tiedostojärjestelmän parametrimuutokset, jotka edellyttävät automaattista konversiota:

- alustusarvon muutos
- tiedostonumeron muutos
- maksimitietuemäärän muutos
- hajautustaulun koon muutos
- moniulotteisen hajautustaulun dimensioiden muutokset

Tietokannan toteutusmäärittelyn muutokset, jotka edellyttävät automaattista konversiota:

- informaatiota hukkaamattomat objektirelaation riippuvuustyypin muutokset
- objektirelaation talletusrakenteen muutos

Informaatiota hukkaamattomiin objektirelaatioiden riippuvuustyyppien muutoksiin kuuluvat mm. riippuvuustyypiltään funktionaalisen objektirelaation muuttaminen monimääräääväksi sekä yksisuuntaisen objektirelaation konvertointi kaksisuuntaiseksi. Toisinpäin konvertointi ei aina ole mahdollista. Esimerkiksi monimääräävän objektirelaation konversiossa funktionaaliseksi menetettäisiin osa monikoista. Säilytettävien monikkojen valintaa ei voi jättää automaattisten toimenpiteiden vastuulle.

6.8. POHJAUTUMINEN TOISEEN

Objektikokoelman, -hakemiston, -relaation tai laskennallisen suhteen pohjautuminen toiseen on automaattisen konversion erityistapaus. Tyypillisesti näin voi käydä nimen muuttuessa.

Alustusohjelmissa ei automaattisia toimenpiteitä tehdä. Tästä syystä tällä lauseella ei ole merkitystä alustusohjelmien kuvauksissa.

Objektikokoelman, -hakemiston, -relaation tai laskennallisen suhteen pohjautuessa toiseen on sekä lähde- että kohdetietokannan ilmentymien nimet esiteltävä. Tietokantatunnisteiden käyttö ei ole tarpeellista, sillä lauseyhteydestä voidaan päätellä tarkoitetaanko objektikokoelmalla, -hakemistolla, -relaatiolla tai laskennallisella suhteella lähde- vai kohdetietokannan ilmentymää.

basedOnDefinition ::=
    CONVERT *basedOnDeclaration* [{ , *basedOnDeclaration* }]
    END *end*

basedOnDeclaration ::=
    destinationIdentifier **FROM** *sourceIdentifier*

Esimerkiksi objektihakemiston interface_by_stage_and_module ja objektirelaation pius_of nimen muutos:

```
CONVERT
        interface_by_position FROM interface_by_stage_and_module,
        plug_ins_of FROM pius_of
END
```

6.9. KONVERSIOLOHKO

Alustus- tai konversiotehtävän ohjelmointi muistuttaa ulkoisesti tietokantaproseduurin ohjelmointia DML:llä. Käytössä ovat DML:n tapaan lausekelauseet, hyppylauseet, objektikokoelman valitsin- ja toistolauseet, kokonaislukuvälin toistolauseet ja ehtolauseet. Lisäksi käytettävissä on poistumislause, jolla konversio voidaan keskeyttää. DML:ään verrattuna pois jäävät perumislause, objektimuuttujan tuhoamislause sekä objektin ja monikon poisto-operaattorit, joille ei ole konversiokuvauksessa mielekästä vastinetta. Konversioissa luodaan aina uusia objektien ja monikoiden ilmentymiä kohdetietokantaan. Jos jotakin objektia tai objektimonikkoa ei kohdetietokannassa tarvita, sitä ei sinne alunperin luodakaan.

conversionBlockDefinition ::=
    CONVERT *destinationIdentifierList body* **END** *end*

body ::=
    [{ *variableDeclaration end* }] [{ *statement end* }]

variableDeclaration ::=
    variableIdentifierList explicitTypeIdentifier
    | *variableIdentifierList explicitTypeIdentifier* = *constantExpression*

explicitTypeIdentifier ::=
    explicitIdentifier

explicitIdentifier ::=
    databaseIdentifier . **identifier**
    | **identifier**

statement ::=
    expressionStatement
    | *collectionSelectorStatement*
    | *collectionIteratorStatement*
    | *subrangeIteratorStatement*
    | *conditionalStatement*
    | *breakStatement*
    | *exitStatement*

Konversiokuvauksen ohjelmoinnissa keskeinen käsite on tietokantatunniste, jolla voidaan ilmaista tarkoitetaanko tietotyypillä, vakiolla, objektikokoelmalla, -hakemistolla, -relaatiolla tai laskennallisella suhteella lähde- vai kohdetietokantaan liittyvää ilmentymää. Tietokantatunniste voidaan antaa varsinaisen tunnisteen nimen etuliitteenä pisteellä erotettuna (produktio *explicitIdentifier*). Mikäli tietokantatunnistetta ei anneta, käytetään oletusarvoisesti kohdetietokannan ilmentymää.

6.9.1. Lausekelause

Lausekelauseet sisältävät sijoituslauseet, funktiokutsut ja joukon päivityslauseet. Sijoituslauseissa skalaari- tai tietuemuuttujan arvoksi asetetaan lausekkeen arvo. Joukon päivityslauseissa objektihakemistoon lisätään objekti. Vastaavasti objektirelaatioihin tai laskennallisiin suhteisiin lisätään monikoita joukon päivityslauseilla.

expressionStatement ::=
    postfixId **:=** *expression*
    | *postfixId* **(** [*expressionList*] **)**
    | *setIdentifier* **:+=** *setItemIdentifier*

postfixId ::=
    identifier
    | *postfixId* **(** *expressionList* **)**
    | *postfixId* **.** **identifier**

setItem ::=
    tuple | *objectVariable*

tuple ::=
    << *objectVariable* [{ **,** *objectVariable* }] [**;** *postfixExpr*] **>>**

6.9.2. Objektikokoelman valitsinlause

Objektikokoelman valitsinlauseella voidaan luoda uusi objekti objektikokoelmaan tai hakea jokin olemassaolevista objekteista indeksin tai hakuavaimen avulla. Lauseen alussa valitsimena käytetyt objektimuuttujat sidotaan objektikokoelman objekteihin. Sidonnan jälkeen suoritetaan joko DO- tai OTHERWISE-sanalla alkavan lohkon lauseet. DO-lohko suoritetaan objektimuuttujien sidonnan onnistuessa ja OTHERWISE-lohko yhdenkin sidonnan epäonnistuessa .

collectionSelectorStatement ::=
    WITH *objectVariableDeclaration* [{ , *objectVariableDeclaration* }]
    [**DO** *body*]
    [**OTHERWISE** *body*]
    END *end*

objectVariableDeclaration ::=
    A NEW *objectVariable* **IN** *explicitCollectionIdentifier* [*indexPosition*]
    | **THE** *objectVariable* **IN** *explicitCollectionIdentifier positionOrKey*

objectVariable ::=
    identifier

indexPosition ::=
    AT *addExpr*

positionOrKey ::=
    indexPosition
    | *searchKey*

searchKey ::=
    (**<<** *expressionList* **>>**)

6.9.3. Objektikokoelman toistolause

Objektikokoelman toistolauseella voidaan hakea objektikokoelmasta joukko objekteja ja suorittaa niille toistolauseen rungon lauseet. Lauseen alussa toistomuuttujat sidotaan objektikokoelmiin tai laskennallisiin suhteisiin. Käsiteltävien objektien määrää voidaan lisäksi rajata valintaehdoilla. Jos kaikkien toistomuuttujien sidonta onnistuu, suoritetaan

mahdollisen DO-lohkon lauseet jokaisella toistokerralla. Jos sidonta epäonnistuu jollekin toistomuuttujalle, suoritetaan mahdollisen OTHERWISE-lohkon lauseet kerran.

SOME-kvanttorilla valitaan vain jokin valintaehdon täyttävistä objekteista. EACH-kvanttorilla käydään läpi kaikki valintaehdon täyttävät objektikokoelman objektit tai laskennalisen suhteen monikot. Valintaehtoina voidaan käyttää hakuavainta tai laskennallisen suhteen monikkoa. Valintaehdon täyttävät objektit haetaan ennen DO-lohkon suoritusta. Valintaehdon muuttujien päivityksellä DO-lohkossa ei siten ole mitään vaikutusta objektien valintaan.

collectionIteratorStatement ::=
FOR *rangeVariable* [{ , *rangeVariable* }] [: *expression*]
[**DO** *body*]
[**OTHERWISE** *body*]
END *end*

rangeVariable ::=
quantifier objectVariable collectionOrCompRelation

collectionOrCompRelation ::=
IN *explicitCollectionIdentifier* [*searchKey*]
| **SUCH THAT** << *objectVariableList* ; *attributeIdentifier* >> **IN**
explicitCompRelationIdentifier

quantifier ::=
EACH | **SOME**

6.9.4. Kokonaislukuvälin toistolause

Kokonaislukuvälin toistolauseessa toistomuuttuja käy läpi arvot alkuarvosta loppuarvoon (päätepisteet mukaan luettuina). Oletusarvoinen askelpituus on 1. Jokaisella toistokerralla suoritetaan DO-lohkon lauseet.

integerSubrangeIterationStatement ::=
FOR *iteratorVariable* **FROM** *addExpr* **TO** *addExpr* [**BY** *addExpr*]
DO *body* **END** *end*

6.9.5. Hyppylause

Hyppylausetta voidaan käyttää toistolauseiden yhteydessä DO-lohkosta poistumiseen. Hyppylauseen jälkeen ohjelman suoritus jatkuu toistolausetta seuraavasta käskystä.

breakStatement ::=
BREAK

6.9.6. Ehtolause

Ehtolause muodostuu yhdestä tai useammasta ehto-osasta ja siihen liittyvästä suorituslohkosta sekä mahdollista lopetuslohkosta. Suorituslohkoista toteutetaan se, jonka ehto ensimmäisenä täyttyy. Jos mikään ehdoista täyty, suoritetaan lopetuslohko.

conditionalStatement ::=
IF *expression* **THEN** *body*
[{ **ELSEIF** *expression* **THEN** *body* }]
[**ELSE** *body*]
END;

6.9.7. Poistumislause

Poistumislausetta voidaan käyttää konversio-ohjelman suorituksen keskeytykseen. Sitä tulee käyttää vain vakavien virhetilanteiden yhteydessä. Normaalitilanteessa konversio-ohjelmista poistutaan, kun kaikki konversiotoimenpiteet on suoritettu.

exitStatement ::=
EXIT

7. TDLCON-ESIKÄÄNTÄJÄ

7.1. YLEISTÄ

Tietokantakonversioiden kuvauskielen määrittelyn valmistuttua aloitettiin kieltä ymmärtävän esikääntäjän suunnittelutyö. TDLCONiksi nimetty esikääntäjä on työkalu, jolla konversiokuvauksista tuotetaan konversio-ohjelmia.

7.2. TAVOITTEET

TDLCON-esikääntäjän tavoitteet jakautuvat kahteen ryhmään: esikääntäjän ja sillä tuotettujen ohjelmien tavoitteisiin. Esikääntäjän tavoitteisiin kuuluvat siirrettävyyteen, käyttöliittymään ja virheiden käsittelyyn liittyvät seikat. Näiden lisäksi TDLCONilta edellytetään luonnollisesti kaikkien CDL-kielen piirteiden ja kuvaustapojen ymmärtämistä.

Siirrettävyys on otettava huomioon TDLCONin suunnittelussa, sillä sen käyttöympäristöinä tullaan käyttämään ainakin VMS- ja UNIX-ympäristöjä. Varsin todennäköisesti tarvitaan myöhemmin myös MS-DOS -ympäristössä toimivaa versiota.

Käyttöliittymän tulee olla yksinkertainen ja selkeä. Esikääntäjän luonne ja käyttötapa edellyttävät, että käyttöliittymä on komentorivipohjainen. Käyttöliittymän tulee lisäksi tarjota lyhyt opasteteksti esikääntäjän käyttötavasta ja tarkoituksesta.

TDLCONin tulee havaita syötetiedostoissaan esiintyvät virheet ja ilmoittaa niistä käyttäjälle. Virhetilanteet on jaettava vakavuutensa puolesta kahteen luokkaan: varoituksiin ja virheisiin. Varoitukset sisältävät lieviä kuvauskielen käyttövirheitä tai epäilyttäviä ohjelmointitapoja, jotka eivät estä koodin tuottoa. Virheet sisältävät vakavammat syntaksi- tai käyttövirheet, jotka estävät aina koodin tuoton.

Konversiokuvauksen esikäännös ei saa pysähtyä ensimmäiseen virheeseen, vaan siitä on toivuttava ja etsittävä mahdolliset muut kuvaustiedostossa esiintyvät virheet samalla esikäännöksellä.

Tuotettujen alustus- ja konversio-ohjelmien vaatimuksiin kuuluu niiden koodaustapa. Koska TDLCON on esikääntäjä, ei sen ole tarkoitus tuottaa suoraan objektikoodia. CDL-kielisestä kuvaustiedostosta esikäännetään TDLCONilla C-kielisiä alustus- ja

konversio-ohjelmia, joista edelleen käännetään ja linkataan suorituskelpoisia ohjelmia. C-kielinen ohjelmakoodi on objektikoodiin rinnastettavissa oleva välitulos. Sitä ei säilytetä, vaan se voidaan tuhota suorituskelpoisen ohjelman valmistuttua.

Toinen tuotettujen ohjelmien vaatimuksista liittyy niiden suorituskykyyn. Tietokannat voivat olla varsin suuria ja niille suoritettavat toimenpiteet aikaavieviä. Tästä syystä tuotetuilta alustus- ja konversio-ohjelmilta edellytetään hyvää suorituskykyä. TDLCONin on painotettava tuotettavassa koodissa suorituskykynäkökohtia.

7.3. TOTEUTUS

7.3.1. Ympäristö

TDLCONin kehitysympäristön vaihtoehtoja olivat käyttöympäristöiksi kaavaillut VMS, MS-DOS ja UNIX. Näistä kehitysympäristöksi valittiin UNIX. Valintaan vaikuttivat lähinnä UNIXin tarjoamat tuotekehitystyökalut ja muiden samantyyppisten työkalujen kehityksestä saadut kokemukset.

7.3.2. Työkalut

Tuotekehityksen eri vaiheisiin tarjoaa kehitysympäristö useita työkaluja. Määrittely- ja ohjelmointivaiheissa käytetään flex- ja byacc-työkaluja. Flex on ohjelmakoodin leksikaalisen analysaattorin tuottamiseen tarkoitettu työkalu. Byacc puolestaan on työkalu ohjelmakoodin jäsentäjän tuottamiseen. Ne ovat molemmat yleisesti ohjelmointikielten kääntäjien kehitystyössä käytettyjä työkaluja. [19]

Varsinaisessa ohjelmoinnissa käytetään Gnu-C -kääntäjää ja gmake-työkalua. Gnu-C on ANSI-C -standardin mukainen C-kielen kääntäjä. Gmake puolestaan on ohjelman käännöstyötä helpottava työkalu. Sen avulla voidaan yhdellä komennolla kääntää kaikki kääntämistä tarvitsevat ohjelmamoduulit ja linkata ne suoritusvalmiiksi ohjelmaksi.

Versionhallintaan käytetään RCS-työkalua ja siihen liittyviä aputyökaluja. RCS:n avulla ohjelmamoduulien muutostyöt tehdään hallitusti ja vähällä hallinnollisella vaivalla.

7.3.3. Toteutusvaiheet

Flex- ja byacc-työkalujen tuottama leksikaalisen analysaattorin ja jäsentäjän koodi on C-kielistä. Tästä syystä on luontevaa käyttää TDLCONin toteutuskielenä C-kieltä. Toteutustyö jakautuu neljään osavaiheeseen esikääntäjän toiminnan mukaisesti. Näitä vaiheita ovat:

- leksikaalinen analyysi
- jäsennys
- semanttinen analyysi
- koodin generointi

Leksikaalisen analyysin tehtävä on lukea on syötteenä saamansa tietovirta ja tunnistaa sieltä kieleen kuuluvat tunnisteet ja symbolit. Sen tuloksena esikäännettävästä konversiokuvauksesta on muodostettu peräkkäisten tunnisteiden jono. Tässä toteutusvaiheessa käytetään flex-työkalua leksikaalisen analyysin toteuttavan ohjelmakoodin generointiin.

Jäsennys tarkoittaa leksikaalisen analyysin tuottamien tunnisteiden jäsennystä kuvauskielen syntaktisiksi lauseiksi. Esikäännettävästä konversiokuvauksesta muodostetaan tässä vaiheessa syntaksipuu. Kuvauskielen jäsentäjä toteutetaan byacc-työkalulla, jolle kuvauskielen syntaksi kuvataan BNF-esitystapaa muistuttavalla tavalla. Byacc-työkalu tuottaa kuvauksen perusteella jäsentäjän ohjelmakoodin.

Semanttisessa analyysissa tarkistetaan esikäännettävän kuvauskielen semantiikka. Tärkeän osan semanttista analyysia muodostaa tyypintarkistus, jossa tarkistetaan jokaisen konversiokuvauksen operaattorin operandien tietotyyppien yhteensopivuus ja sallittavuus. Semanttisen analyysin rutiinit lisätään byacc-työkalun jäsennyskuvaustiedostoon.

Koodin generointi on esikäännöksen viimeinen vaihe, jossa tuotetaan syntaksipuusta ohjelmakoodi. TDLCONin koodigeneraattorin tuottama ohjelmakoodi on C-kielistä, joten muuttujien muistipaikkojen ja prosessorin rekisterien käytöstä ei tarvitse huolehtia. Ne ovat vasta C-kielen kääntäjän koodigeneraattorin päätettävissä.

7.4. TOIMINTA

7.4.1. Käyttöliittymä

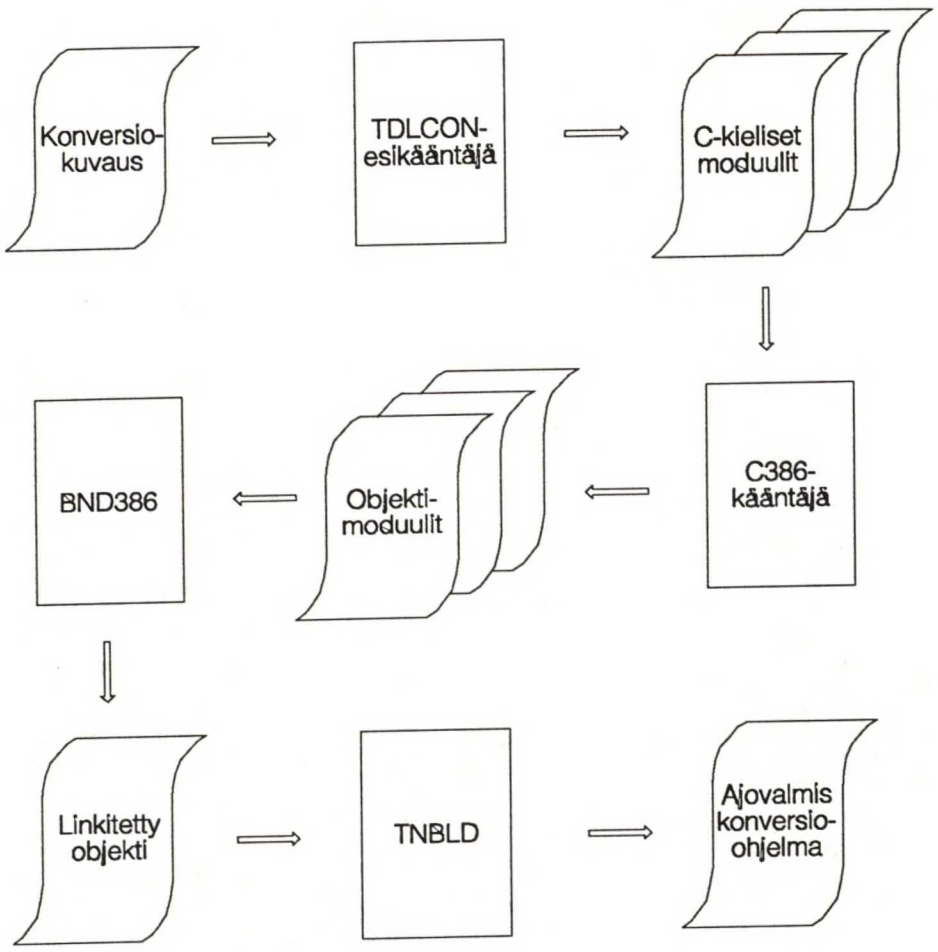
Esikääntäjän käyttöliittymänä on komentorivikäyttöliittymä. Parametrien esitystapa on aina samanlainen riippumatta siitä, minkä käyttöjärjestelmän alaisuudessa ohjelmaa käytetään. Komentosyntaksi on seuraava:

TDLCON [-e virhetiedosto] kuvaustiedosto

Hakasuluissa esitetyt parametrit ovat vapaaehtoisia. Ilman parametreja ajettuna esikääntäjä tulostaa lyhyen käyttöopasteen. Lipulla -e virhetulostukset voidaan ohjata oletusarvoiselta virhetulostuslaitteelta käyttäjän nimeämään tiedostoon. Kuvaustiedosto sisältää alustus- tai konversio-ohjelman CDL-kielisen kuvauksen.

7.4.2. Käännösprosessi

Alustus- ja konversio-ohjelmien käännösprosessi alkaa TDLCONilla suoritettavalla esikäännöksellä. TDLCON lukee CDL-kielisen kuvaustiedoston ja sen kautta edelleen lähde- ja kohdetietokantojen toteutusmäärittelyt (FDL) ja tietokantakaaviot (DDL). Esikäännöksen tuloksena saadaan joukko C-kielisiä ohjelmamoduuleja, jotka yhdessä muodostavat alustus- tai konversio-ohjelman. Ohjelmamoduulit käännetään C-kielen kääntäjällä objektimoduuleiksi ja linkataan yhteen. Lopuksi linkattu objektimoduuli muokataan TNBLD-työkalulla DX 200 -järjestelmän ohjelmalohkoksi.



Kuva 5. Konversio-ohjelman käännösprosessi

7.4.3. Tulostiedostot

TDLCONin tulostiedostoina tuottamat C-kieliset moduulit noudattavat lähteessä [18] kuvattua ANSI-C -standardia. Moduulit ovat siten periaatteessa käännettävissä millä tahansa ANSI-C -standardin mukaisella kääntäjällä.

Tuotetut konversio-ohjelmat käyttävät hyväkseen FIXLIB- ja DEACON-kirjastojen tarjoamia palveluja tiedostojen käsittelyyn. Kirjastorajapintojen muutoksilla on siten suora vaikutus myös TDLCONiin.

Hyvä suorityskyky oli yksi tuotetuilta ohjelmilta vaadituista ominaisuuksista. Levytiedostoja konvertoitaessa pullonkaulaksi muodostuu levylle suoritettavat luku- ja kirjoitustehtävät. Suorituskykyä parantaakseen TDLCON pyrkii minimoimaan levyyksikössä hitautta aiheuttavan luku- ja kirjoituspään liikkumisen. Minimointi tapahtuu paikallistamalla konvertoitavien tiedostojen käsittelyä pienille alueille kerrallaan. [20]

Toinen suorituskykyä parantava tekijä olisi konversion suoritus keskusmuistitiedostoja hyväksi käyttäen. FIXLIB-kirjasto ei tällä hetkellä tue tälläistä ominaisuutta. Jatkokehityskohteena keskusmuistitiedostoihin kohdistuva konversio on kuitenkin huomionarvoinen vaihtoehto.

8. YHTEENVETO

DX 200 -järjestelmän puhelinkeskuksen ominaisuudet toteuttava ohjelmisto muodostaa ohjelmistopaketin. Uusien ominaisuuksien lisääminen puhelinkeskukseen voidaan toteuttaa ohjelmistopakettia vaihtamalla. Ohjelmistopaketin vaihdon yhteydessä on puhelinkeskuksen tiedostoihin ja tietokantoihin talletettu tieto siirrettävä uuden ohjelmistopaketin tiedostoihin ja tietokantoihin. Mikäli uuden ja vanhan ohjelmistopaketin tiedosto- ja tietokantarakenteet ovat erilaisia, täytyy tieto konvertoida vanhasta talletustavasta uuteen.

Konversiot toteutetaan konversio-ohjelmilla. Ne ovat jotakin tiettyä ohjelmistopaketinvaihtoa silmälläpitäen ohjelmoituja ohjelmia. Niiden ohjelmointikielinä on käytetty C- ja PL/M-kieliä.

Tietokantakonversioita varten on diplomityönä kehitetty konversio-ohjelmien kuvauskieli. Sen avulla voidaan konversio-ohjelmat kuvata korkeammalla abstraktiotasolla kuin C- ja PL/M-kielillä.

Kuvauskieli valittiin kahden vaihtoehdon väliltä. Ensimmäinen vaihtoehto oli deklaratiivinen kuvauskieli, jossa konversiotoimenpiteet oli ennalta määritelty ja sisällytetty kuvauskielen syntaksiin. Sen hyvänä puolena oli lyhyt ilmaisumuoto. Lisäksi ennalta määritellyistä konversiotoimenpiteistä johtuen, sillä kuvatut konversio-ohjelmat olisi voitu toteuttaa tehokkaasti. Haittapuolina sillä olivat hankalasti toteutettavissa olevat kerrannaisvaikutukset, joita syntyy objektikokoelmia konvertoitaessa. Pahimmillaan objektikokoelman konversio voisi edellyttää useiden objektihakemistojen, -relaatioiden ja laskennallisten suhteiden konversiotarpeen.

Toinen kuvauskielen vaihtoehto oli proseduraalinen kieli. Siinä konversiotoimenpiteitä ei ole etukäteen rajattu, vaan käyttäjä voi ohjelmoida konversiolohkojen avulla kulloinkin tarvittavat konversiotoimenpiteet. Proseduraalisen kielen lauseet ovat suurelta osin samoja kuin DML-kieleen sisältyvät lauseet. Sen hyvänä puolena on ilmaisuvoiman rikkaus, jonka ansiosta sillä voidaan kuvata monimutkaisetkin konversiotoimenpiteet. Toinen merkittävä etu kuvauskieltä käyttöönotettaessa on sen samankaltaisuus DML:n kanssa. Käyttöönottokynnys on samojen käsitteiden ja lauserakenteiden ansiosta huomattavasti alhaisempi kuin kokonaan uuden esitystavan tapauksessa. Haittapuolena on deklaratiiviseen kuvauskieleen verrattuna laveampi esitystapa. Lisäksi proseduraalisen kuvauskielen käyttäjältä vaaditaan hieman enemmän ohjelmointitaitoa.

Tietokantakonversio-ohjelmien kuvauskieleksi valittiin proseduraalinen kuvauskieli. Valintaan vaikuttivat edellämainituista tekijöistä erityisesti deklaratiivisen kielen vaikeasti toteutettavat kerrannaisvaikutukset sekä proseduraalisen kielen laaja ilmaisuvoima. Lisäksi sen yhtenevyys TDL-kieleen sisältyvään DML-kielen kanssa alentaa käyttöönottokynnystä ja vähentää käyttäjille suunnattavan koulutuksen tarvetta.

Diplomityö sisälsi myös kuvauskielen toteuttavan esikääntäjän laadinnan. Sen avulla kuvauskielellä laadituista konversiokuvauksista esikäännetään C-kielisiä konversio-ohjelmia. Esikääntäjän käyttöympäristöinä tulevat olemaan ainakin VMS ja UNIX. Myös MS-DOS versiota tarvitaan todennäköisesti tulevaisuudessa. Useasta käyttöympäristöstä johtuen esikääntäjän tärkeimpiä vaatimuksia on siirrettävyys. Toinen tärkeä vaatimus liittyy esikääntäjällä tuotettuihin konversio-ohjelmiin, joiden suorituskykyä parantaakseen esikääntäjän on pyrittävä optimoimaan tuotettavaa koodia. Tärkein suorituskykyyn vaikuttava optimointitekijä on levy-yksikön lukupään liikkumisen minimointi, joka voidaan toteuttaa paikallistamalla konversioiden suoritus pienille tietokantatiedostojen alueille kerrallaan.

9. JOHTOPÄÄTÖKSET

Kehitetyn kuvauskielen ja esikääntäjän käyttökelpoisuus selviää lopullisesti vasta käytössä. Huolellisen määrittelyn ja suunnittelun uskoisin kuitenkin kantavan satoa ja uuden kuvaustavan osoittautuvan edistysaskeleeksi konversio-ohjelmien laadintatyössä.

Kuvauskieli ei ole tässä dokumentissa esitetyssä muodossaan missään nimessä lopullinen. Jatkokehitystarpeita ilmenee varmasti käyttöönoton jälkeen useita. Mahdollinen jatkokehityshanke on esimerkiksi tiedostoista tietokantaan konversioiden kuvausmahdollisuus. Myös TDL-kielen ja monitietokantajärjestelmän jatkokehitys edellyttävät kuvauskielen ylläpitoa.

TDLCON-esikääntäjän jatkokehitykseen on myös varauduttava. Kuvauskielen syntaksin muuttuessa on luonnollisesti myös esikääntäjää päivitettävä. Tämän lisäksi esikääntäjän jatkokehitysnäkymiin kuuluvat erilaiset konversio-ohjelmien suorituskykyvyn parantamiseen liittyvät tekijät, kuten konversioiden suoritus keskusmuistitiedostoja käyttäen.

VIITELUETTELO

1. Hjort, P. Tietotyyppien määrittely ja toteuttaminen SDL-kuvauskielessä. Diplomityö, TKK 1990. 56 s.

2. Ruohtula, E. SDL-kuvauskielen käyttö ohjelmoinnissa. Diplomityö, TKK 1991. 91 s.

3. Tikkanen, M. TDL-tietokantakieli. Espoo 1992, Nokia Telecommunications Oy. 102 s.

4. Räsänen, P. MML-ohjelmien suunnitteluohje. Espoo 1990, Nokia Telecommunication Oy. 83 s.

5. Liuhto, A. DX 200 tiedostopalvelut. Espoo 1991, Nokia Telecommunications Oy. Toimintoluokan kuvaus. 23 s.

6. Haukilahti, J., Törnqvist, J. Prosessiperheen suunnittelijan käsikirja. Espoo 1992, Nokia Telecommunications Oy. 43 s.

7. Taanila, S. Paketointi, versioituminen ja nimeäminen F5:ssä. Espoo 1990 Nokia Telecommunications Oy. 32 s.

8. Karttunen, M. Tietokannan hallinta. Espoo 1991, Nokia Telecommunications Oy. Palvelulohkon kuvaus. 37 s.

9. Hintsala, E. DX 200 ohjelmistokonfiguraation hallinta. Espoo 1992, Nokia Telecommunications Oy. Käyttökäsikirja. 50 s.

10. Liuhto, A. Tiedostojen konversio-ohjelmien laatiminen. Espoo 1992, Nokia Telecommunications Oy. 21 s.

11. Liuhto, A. Tiedostokonversioiden tukikirjasto FIXLIB. Espoo 1993, Nokia Telecommunications Oy. 39 s.

12. Liuhto, A. DX 200 tiedostokonversion hallinta. Espoo 1992, Nokia Telecommunications Oy. Käyttökäsikirja, 4 s.

13. Horowitz, E. Fundamentals of Programming Languages. 2. p. Berlin 1984, Springer-Verlag. 446 s.

14. Meetham, A.R. Encyclopaedia of linguistics, information and control. Lontoo 1969, Pergamon Press Ltd. 718 s.

15. Korpela, J., Larmela, T., Planman, A. Pascal-ohjelmointikieli. 2. p., Espoo 1980, OtaDATA ry. 362 s.

16. Gries, D. The Science of Programming. New York 1981, Springer-Verlag. 366 s.

17. Lindqvist, M., Ruohtula, E., Kettunen, E., Tuominen H. The TNSDL Book. Espoo 1992, Nokia Telecommunications Oy. 222 s.

18. ANSI X3.159. Programming language - C. American National Standards Institute, Inc, 1989. 219 s.

19. Aho, A.V., Sethi, R., Ullman, J.D. Compilers: Principles, Techniques, and Tools. Reading 1986, Addison-Wesley. 796 s.

20. Aarnio, I. TDLCON-vaatimusmäärittely. Espoo 1993, Nokia Telecommunications Oy. 23 s.

LIITE 1: SYNTAKSI

definition ::=
    constantDefinition
    | *typeDefinition*
    | *syncServiceDefinition*

constantDefinition ::=
    CONSTANT { *constantDeclaration* }

constantDeclaration ::=
    constantIdentifier **=** *constantExpression end*

constantExpression ::=
    expression

typeDefinition ::=
    [**ABSTRACT**] **TYPE** *typeIdentifier*
        [*constantDefinition*]
        REPRESENTATION *typeDeclaration end*
        [**OPERATORS** { *operatorDefinition* }]
    ENDTYPE [*typeIdentifier*] *end*

typeDeclaration ::=
    typeIdentifier
    | *integerSubrangeDeclaration*
    | *enumarationDeclaration*
    | *bitsDeclaration*
    | *arrayDeclaration*
    | *structDeclaration*
    | *bitStructDeclaration*
    | *unionDeclaration*
    | *pointerDeclaration*

integerSubrangeDeclaration ::=
    (*constantExpression* **:** *constantExpression* **)**

enumerationDeclaration ::=
    ENUM

enumConstantDeclaration [{ , *enumConstantDeclaration* }]
ENDENUM

enumConstantDeclaration ::=
constantIdentifier [= *constantDeclaration*]

bitsDeclaration ::=
BITS (*constantExpression*)

arrayDeclaration ::=
ARRAY { *dimension* } **OF** *typeDeclaration*

dimension ::=
(*constantExpression*)

structDeclaration ::=
[**PACKED**] **STRUCT** { *fieldDeclaration* } **ENDSTRUCT**

fieldDeclaration ::=
fieldIdentifier typeDeclaration end

bitStructDeclaration ::=
BITSTRUCT { *bitFieldDeclaration* } **ENDBITSTRUCT**

bitFieldDeclaration ::=
fieldIdentifier constantExpression end

unionDeclaration ::=
UNION { *fieldDeclaration* } **ENDUNION**

pointerDeclaration ::=
POINTER [**NEAR** | **FAR**] (*typeIdentifier*)

syncServiceDefinition ::=
SERVICES SYNC { *syncServiceDeclaration* }

syncServiceDeclaration ::=
operatorDefinition
| *viewedDefinition*
| *libraryDefinition*

operatorDefinition ::=
[**PROCEDURE**] *operatorIdentifier* **(** [*formalParameterList*] **)**
-> [*typeIdentifier*] [**,** *implementationDefinition*] *end*

formalParameterList ::=
formalParameter [{ **,** *formalParameter* }]

formalParameter ::=
passingMethod parameterIdentifier typeIdentifier

passingMethod ::=
IN
| **IN / OUT** [**NEAR** | **FAR**]

implementationDefinition ::=
[**NEAR** | **FAR**] [**=>** | **<=**]

viewedDefinition ::=
VIEWED [**NEAR** | **FAR**] *viewedDeclaration* [{ **,** *viewedDeclaration* }] *end*

viewedDeclaration ::=
viewedIdentifier [{ **,** *viewedIdentifier* }] *typeDeclaration*

libraryDefinition ::=
LIBRARY *libraryIdentifier end*
{ *operatorDefinition* | *viewedDefinition* }
ENDLIBRARY *libraryIdentifier end*

conversionProgramDefinition ::=
CONVERSION *conversionIdentifier*
[**FROM** *sourceIdentifier databaseAttributes*]
TO *destinationIdentifier databaseAttributes*
[{ *definition* }]
[{ *conversionDefinifion* }]
END [*conversionIdentifier*] *end*

databaseAttributes ::=
DATABASE : *databaseIdentifier end*
FILE : *fileIdentifier end*
DIRECTORIES : *directoryIdentifier* [{ **,** *directoryIdentifier* }] *end*

conversionDefinition ::=
excludeDefinition
| *basedOnDefinition*
| *conversionBlockDefinition*

excludeDefinition ::=
SKIP *excludeIdentifier* [{ , *excludeIdentifier* }] **END** *end*

basedOnDefinition ::=
CONVERT *basedOnDeclaration* [{ , *basedOnDeclaration* }]
END *end*

basedOnDeclaration ::=
destinationIdentifier **FROM** *sourceIdentifier*

conversionBlockDefinition ::=
CONVERT *destinationIdentifier* [{ , *destinationIdentifier* }] *body* **END** *end*

body ::=
[{ *variableDeclaration end* }] [{ *statement end* }]

variableDeclaration ::=
variableIdentifierList explicitTypeIdentifier
| *variableIdentifierList explicitTypeIdentifier* = *constantExpression*

explicitTypeIdentifier ::=
explicitIdentifier

explicitIdentifier ::=
databaseIdentifier . **identifier**
| **identifier**

statement ::=
expressionStatement
| *collectionSelectorStatement*
| *collectionIteratorStatement*
| *subrangeIteratorStatement*
| *conditionalStatement*
| *breakStatement*
| *exitStatement*

expressionStatement ::=
    postfixId **:=** *expression*
    | *postfixId* **(** [*expressionList*] **)**
    | *setIdentifier* **:+=** *setItemIdentifier*

postfixId ::=
    identifier
    | *postfixId* **(** *expressionList* **)**
    | *postfixId* **.** **identifier**

setItem ::=
    tuple | *objectVariable*

tuple ::=
    << *objectVariable* [{ **,** *objectVariable* }] [**;** *postfixExpr*] **>>**

expressionList ::=
    expression [{ **,** *expression* }]

expression ::=
    expression **OR** *andExpr*
    | *andExpr*

andExpr ::=
    andExpr **AND** *compExpr*
    | *compExpr*

compExpr ::=
    tuple **IN** *explicitIdentifier*
    | *addExpr compOp addExpr*
    | *addExpr*

compOp ::=
    IS | **=** | **/=** | **<** | **>** | **<=** | **>=**

addExpr ::=
    addExpr addOp mulExpr
    | *mulExpr*

addOp ::=
    + | −

mulExpr ::=
    mulExpr mulOp unaryExpr
    | *unaryExpr*

mulOp ::=
    * | / | **MOD**

unaryExpr ::=
    unaryOp primaryExpr
    | *primaryExpr*

unaryOp ::=
    + | − | # | @ | **$**

primaryExpr ::=
    postfixExpr
    | **(** *expression* **)**
    | *numericLiteral*
    | **stringLiteral**

collectionSelectorStatement ::=
    WITH *objectVariableDeclaration* [{ **,** *objectVariableDeclaration* }]
    [**DO** *body*]
    [**OTHERWISE** *body*]
    END *end*

objectVariableDeclaration ::=
    A NEW *objectVariable* **IN** *explicitCollectionIdentifier* [*indexPosition*]
    | **THE** *objectVariable* **IN** *explicitCollectionIdentifier positionOrKey*

objectVariable ::=
    identifier

indexPosition ::=
    AT *addExpr*

positionOrKey ::=
    indexPosition
    | *searchKey*

searchKey ::=
    (**<<** *expressionList* **>>** **)**

collectionIteratorStatement ::=
    FOR *rangeVariable* [{ **,** *rangeVariable* }] [**:** *expression*]
    [**DO** *body*]
    [**OTHERWISE** *body*]
    END *end*

rangeVariable ::=
    quantifier objectVariable collectionOrCompRelation

quantifier ::=
    EACH | **SOME**

collectionOrCompRelation ::=
    IN *explicitCollectionIdentifier* [*searchKey*]
    | **SUCH THAT** **<<** *objectVariableList* **;** *attributeIdentifier* **>>** **IN**
    explicitCompRelationIdentifier

integerSubrangeIterationStatement ::=
    FOR *iteratorVariable* **FROM** *addExpr* **TO** *addExpr* [**BY** *addExpr*]
    DO *body* **END** *end*

iteratorVariable ::=
    identifier

breakStatement ::=
    BREAK

conditionalStatement ::=
    IF *expression* **THEN** *body*
    [{ **ELSEIF** *expression* **THEN** *body* }]
    [**ELSE** *body*]
    END;

exitStatement ::=
    EXIT

end ::=
    [**COMMENT stringConstant**] ;

variableIdentifierList ::=
    variableIdentifier [{ , *variableIdentifier* }]

explicitCollectionIdentifier ::=
    explicitIdentifier

explicitCompRelationIdentifier ::=
    explicitIdentifier

constantIdentifier ::=
    identifier

typeIdentifier ::=
    identifier

fieldIdentifier ::=
    identifier

operatorIdentifier ::=
    identifier

parameterIdentifier ::=
    identifier

viewedIdentifier ::=
    identifier

libraryIdentifier ::=
    identifier

conversionIdentifier ::=
    identifier

sourceIdentifier ::=
    identifier

destinationIdentifier ::=
identifier

databaseIdentifier ::=
identifier

fileIdentifier ::=
identifier

directoryIdentifier ::=
identifier

variableIdentifier ::=
identifier

setIdentifier ::=
identifier

attributeIdentifier ::=
identifier

excludeIdentifier ::=
identifier

LIITE 2: ESIMERKKI

Tässä esimerkissä on kuvattu luvussa 3 esitellylle tietokannalle toteutettava konversio. Lähdetietokanta sisältää kaksi objektikokoelmaa: subscribers ja subsc_services. Edelliselle on määritelty objektihakemisto sub_by_sub_no. Lisäksi objektikokoelmien välillä on objektirelaatio services_of.

Kohde- ja lähdetietokannan välillä on kolme eroavaisuutta, jotka edellyttävät konversiota. Ensinnäkin kohdetietokannassa objektikokoelman subscribers objektien tietotyyppiin on lisätty kenttä area_code. Tämä muutos ei näy tietokantakaavoissa, sillä tietotyyppien määrittelyt sisältyvät ympäristömäärittelypakettiin. Toiseksi objektikokoelmaan subscribers liittyvän objektihakemiston hakuavaimeen on lisätty kenttä area_code. Samalla sen nimeä on muutettu. Kolmanneksi ja viimeiseksi objektikokoelmien välisen objektirelaation nimi on muuttunut subsc_services_of:ksi. Lisäksi sen riippuvuustyyppi toiseen suuntaan on muuttunut funktionaalisesta monimääräväksi.

Edellä esitetyistä konversiotarpeista ensimmäinen eli objektikokoelman subscribers tietotyypin muutos vaatii ohjelmoitua konversiolohkoa. Sama pätee sille määritellyn objektihakemiston muutoksen yhteydessä. Koska molemmat konversiot liittyvät samaan objektikokoelmaan, ne on yksinkertaisinta toteuttaa samalla konveriolohkolla. Objektirelaation nimenmuutos edellyttää toiseen tietokantatiedostoon pohjautuvan konversiolauseen käyttöä. Sen riippuvuustyypin muutos on automaattisesti toteutettavissa, joten erillistä konversiolohkoa ei sille tarvitse laatia. Ainoana konversiokuvauksen ulkopuolelle jätetty objektikokoelma subsc_services konvertoidaan automaattisesti. Sen alustusarvo on muuttunut 0:sta 255:een.

Lähde- ja kohdetietokannan tietokantakaaviot ja toteutusmäärittelyt on ensin esitelty. Niitä seuraa konversio-ohjelman kuvaustiedosto. Toteutusmäärittelyt on esitelty yksinkertaistettuina. Niistä on jätetty konversio-ohjelmien kannalta merkityksettömät toteutuskohtaisten vakioiden ja tietokantaproseduurien hyppytaulujen määrittelyt pois esityksen yksinkertaistamiseksi ja lyhentämiseksi.

```
/* Lähdetietokannan tietokantakaavio ja toteutusmäärittely */

DATABASE examdb

        subscribers     ARRAY OF  subscribers_t;
        subsc_services ARRAY OF  subsc_services_t;

        sub_by_sub_no
                DIRECTORY ON subscribers: << subsc_no >>;

        services_of
                RELATION
                        sub  IN subscribers;
                        serv IN subsc_services;
                WHERE sub <-> serv;

ENDDATABASE examdb;


IMPLEMENTATION OF DATABASE examdb

        COLLECTIONS
                subscribers
                        fileNumber = 1;
                        initValue  = 0;
                END subscribers;
                subsc_services
                        fileNumber = 2;
                        initValue  = 0;
                END subsc_services;

        DIRECTORIES
                sub_by_sub_no
                        fileNumber = 3;
                        REPRESENTATION HASH(SUB_HA);
                END sub_by_sub_no;

        RELATIONS
                services_of
                        DEPENDENCY ->
                                fileNumber = 4;
                                REPRESENTATION DENSE;
                        DEPENDENCY <-
                                fileNumber = 5;
                                REPRESENTATION DENSE;
                END services_of;

END examdb;
```

```
/* Kohdetietokannan tietokantakaavio ja toteutusmäärittely */

DATABASE examdb

        subscribers     ARRAY OF  subscribers_t;
        subsc_services ARRAY OF  subsc_services_t;

        sub_by_sub_and_area_no
                DIRECTORY ON subscribers: << subsc_no, area_code >>;

        subsc_services_of
                RELATION
                        sub  IN subscribers;
                        serv  IN subsc_services;
                WHERE sub <->> serv;

ENDDATABASE examdb;


IMPLEMENTATION OF DATABASE examdb

        COLLECTIONS
                subscribers
                        fileNumber = 1;
                        initValue  = 0;
                END subscribers;
                subsc_services
                        fileNumber = 2;
                        initValue  = 0xFF;
                END subsc_services;

        DIRECTORIES
                sub_by_sub_and_area_no
                        fileNumber = 3;
                        REPRESENTATION HASH(SUB_HA);
                END sub_by_sub_and_area_no;

        RELATIONS
                subsc_services_of
                        DEPENDENCY ->>
                                fileNumber = 4;
                                overflowFileNumber = 5;
                                REPRESENTATION DENSE;
                        DEPENDENCY <-
                                fileNumber = 6;
                                REPRESENTATION DENSE;
                END subsc_services_of;

END examdb;
```

```
CONVERSION exafix
        FROM src
                DATABASE: examdb;
                FILE: examdbgx.fdl;
                DIRECTORIES: /F5_ENV/4_8_1/DTY/SDL_FORM;
        TO dst
                DATABASE: examdb;
                FILE: examdbgx.fdl;
                DIRECTORIES: /AARNIO/DTYO/TMP,
                             /F5_ENV/5_3_0/DTY/SDL_FORM;

        /* Määrittelyt */

        SERVICES SYNC
                copy_dir_number(
                        IN/OUT destination dst.subsc_no_t,
                        IN     source      src.subsc_no_t
                ) ->, NEAR =>;
                output_error(
                        IN  text  error_text_t
                ) ->, NEAR =>;

        /* Konversiotoimenpiteiden kuvaukset */

        CONVERT
                basic_services FROM subsc_services
        END;

        CONVERT subscribers, sub_by_sub_and_area_no
                FOR EACH old_sub IN src.subscribers
                DO
                        WITH A NEW sub IN dst.subscribers
                        DO
                                sub.area_code := undefined_c;
                                copy_dir_number(sub.subsc_no,
                                                old_sub.subsc_no);
                                IF old_sub.subsc_type = unk_subsc_c THEN
                                        sub.subsc_type := ext_subsc_c;
                                ELSE
                                        sub.subsc_type := old_sub.subsc_type;
                                END;
                                sub_by_sub_and_area_no :+= sub;
                        OTHERWISE
                                output_error('Subscriber creation failed.');
                                EXIT;
                        END;
                OTHERWISE
                        output_error('No objects in source collection.');
                END;
        END;
END exafix;
```